Landing Throttleable Hybrid Rockets with Hierarchical Reinforcement Learning in a Simulated Environment

Francesco Mikulis-Borsoi* **and Paul Gesel***
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA

Abstract

(Paul)
In this paper, we will develop a hierarchical MDP structure for vertical rocket landing. We decompose the problem into several sub-problems of vertical landing, namely velocity control and vertical stability control. We exploit MDP coupling and symmetrical properties to significantly reduce the size of the state space. We make two contributions in this paper: 1) a graphical framework for defining MDPs in a standardized format and 2) a hierarchical MDP structure that is able to successfully land the rocket within our goal bounds more than 95 percent of the time. We validate our approach by comparing its performance to a baseline RRT search.

Introduction (Francesco)

The objective of this paper is to develop a controller that can land a rocket vertically in a simulated 3D environment. Self-landing rockets have been theorized over the past century, but until 2015, no one had successfully completed a successful vertical landing with thrust vectoring in a real rocket. The controller is required to reach predefined goal conditions from a variety of starting states. A popular approach for vertical rocket landing includes MPC, such as that seen in R. Ferrante (Ferrante 2017), but in most cases, the problem is solved in a 2D environment. We will solve this problem with Reinforcement Learning (RL) under a hierarchical Markov Decision Process (MDP) framework in 3D.

The case for MDPs: a philosophical perspective (Paul)

The policy learned with an MDP formulation is reflexive. That is, the policy directly maps state to action without planing online. Hence, an agent acting in accordance to a learned policy is essentially a reflex agent. We pose the question of whether or not a reflex agent exhibits intelligence. One might ask, "what is intelligence"? According to the Goertzel, the capability of an agent to achieve complicated goals in dynamic environments is a suitable interpretation of intelligence (Goertzel 2009). We take the perspective that intelligence is neither reflex nor planning, but a combination of both. Sutton and other researchers of phenomenology, argue that acting appropriately in the heat of the moment is still a form of intelligence (Sutton et al. 2011). This is further reinforced by Krakauer, who argues that human tasks at many levels of expertise are composed of intelligent reflexes and deliberative decisions (Krakauer 2019).

A reflex agent on its own may not be intelligent, but an agent that intentionally develops its reflexes for skills that it will need is brilliant. Arguably, developing and remembering a reflex for a repeated task is critical component of intelligence. Our proposed hierarchical MDP structure aims to capture this phenomenon; the role of reflex in intelligent systems. We implement low level controllers (or MDPs), which resemble the reflexes of an intelligent system. Under the hierarchical MDP structure, both deliberate decision making and intelligent reflexes can be modelled.

The case for MDPs: a practical perspective (Paul)

A practical issue of online planning is encountered when edge evaluations or node expansions are computationally expensive. Search methods, such as Lazy Weighted A* (Cohen, Phillips, and Likhachev 2014), aim to alleviate this problem by postponing expensive computations until they are absolutely necessary. Even a computationally aware planner must ultimately perform full edge evaluations. Complex physics problems frequently require massive computational power. For example, simulating a rocket engine in an optimized computational fluid dynamics library can require several days of CPU time (Invigorito, Cardillo, and Ranuzzi 2014). This effectively eliminates online search as an option in a real-time setting. Other practical benefits of direct state-action mapping includes ease of implementation on low level hardware and low computational cost.

Notation (Paul)

We will use the notation below throughout the paper. Notably, elements of vectors are indexed with a single subscript, where the subscript of x, y, or z indicate the first, second, or third component of a vector, respectively. Some symbols are written with superscript notation. This is only for designation and does not represent a mathematical op-

*These authors contributed equally to this work.

eration. For example, the the superscript r in R^r designates a that R is the rotation matrix from the rocket's coordinates to the global coordinates. Furthermore, the dot ($\dot{}$) notation represents a time derivative.

t: time
x: position along x axis in global coordinates
y: position along y axis in global coordinates
z: position along z axis in global coordinates
$\dot{x}$: velocity along x axis in global coordinates
$\dot{y}$: velocity along y axis in global coordinates
$\dot{z}$: velocity along z axis in global coordinates
ω: angular velocity in global coordinates
u: control vector
θ_x: gimbal angle about its x axis
θ_y: gimbal angle about its y axis
γ_x: rocket angle about its x axis
γ_y: rocket angle about its y axis
F: force applied to the rocket in global coordinates
T: torque applied to the rocket in global coordinates
$m(t)$: non-throttled rocket thrust force
g: force of gravity on rocket
$f(x, w)$: unknown forces and torques
m^r: mass of the rocket
cm: vector from the center of mass to the gimbal in rocket coordinates
l: vector from the center of mass to the gimbal in global coordinates
R^r: rotation matrix from rocket coordinates to global coordinates
d: gimbal direction in rocket coordinates
D: rocket axial direction in global coordinates
a: action
S: state vector
$R(s)$: reward function
$R^t(s)$: terminal reward function

Background

In this section, we will derive relevant theoretical models for thrust vectoring rocket dynamics. We will then give a high level overview of MDPs, coupled MDPs, symmetrical MDP properties, hierarchical MDPs, and finally RRT.

Rocket model (Paul)

In this section, we will derive the dynamics of a throttleable thrust vectoring rocket. Notably, our implementation will rely on an open-source software to simulate the rocket's dynamics. The model derived in this section provides insight on vertical rocket landing, which will help us create reasonable state definitions in the later sections.

Our model has 5 actuators: one to throttle the main thruster, two to control the gimbal angles of the thruster, and two more to set the force of the lateral thrusters. The unthrottled force of the main thruster is denoted $m(t)$. Notably, the rocket is under-actuated and cannot move the gimbal while using lateral thrusters. More specifically, the pair of actions

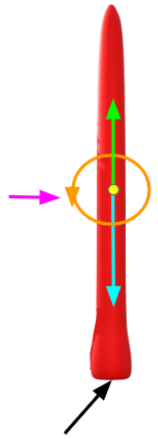

Figure 1: rocket free body diagram

u_2 and u_5 and pair of actions u_3 and u_4 cannot be changed at a single instant. Fig. 1 illustrates a free body diagram with all forces and torques modeled. The black arrow indicates the thrust vectoring force F. Since it is applied at the bottom of the rocket, causing a torque T (shown in orange). The purple arrow indicates the lateral thruster force F^l. This is applied at the center mass of the rocket and perpendicular to the rocket's z axis. The remaining arrows represent gravity g and unknown aerodynamic forces. Since the aerodynamic forces depend on unattainable sensor measurements, such as temperature and wind, we denote those forces and torques as $f(x, w)$, where w is an unobserved variable. We define the rocket's state vector in the following way,

$$S = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \gamma_x, \gamma_y, \omega_x, \omega_y]^T \in \mathbb{R}^{10}$$

where x, y, and z represent the rocket's position, $\dot{x}$, $\dot{y}$, and $\dot{z}$ represent the rocket's velocity, γ_x and γ_y indicate the rocket's pitch and yaw, and ω_x and ω_y are the rocket's angular velocities. The dot notation indicates the time derivatives of each variable. With this definition, the rocket's orientation is expressed as follows.

$$R^r = \begin{bmatrix} \cos(\gamma_x) & 0 & \sin(\gamma_x) \\ \sin(\gamma_y)\sin(\gamma_x) & \cos(\gamma_y) & -\sin(\gamma_y)\cos(\gamma_x) \\ -\cos(\gamma_y)\sin(\gamma_x) & \sin(\gamma_y) & \cos(\gamma_y)\cos(\gamma_x) \end{bmatrix} \quad (1)$$

We do not model roll for two reason 1) none of the modeled forces create a torque about the rocket's axial direction and 2) if the rocket does roll, the rocket has no ability to stop the motion, causing it to lose control. The five available controls are defined as follows.

$$u = [u_1, u_2, u_3, u_4, u_5]^T \in \mathbb{R}^5$$

where u_1 is the throttle multiplier of the main thrust, u_2 and u_3 are the gimbal angles, and u_4 and u_5 are the lateral trust forces in the x and y directions, respectively. Finally, the vector from the rocket's gimbal to its center of mass in global coordinates is express as l and force of gravity on rocket g.

$$l = R^r \begin{bmatrix} cm_x \\ cm_y \\ cm_z \end{bmatrix} \quad (2)$$

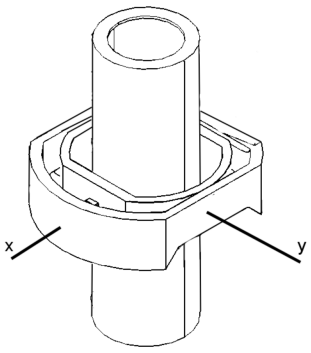

Figure 2: schematic of the gimbal design

$$g = \begin{bmatrix} 0 \\ 0 \\ -9.81m^r \end{bmatrix} \tag{3}$$

Thrust vectoring dynamics (Paul)

The gimbal is composed of two revolute joints; one with its axis of rotation in x and one in y, as shown in Fig. 2. These angles are denoted as θ_x and θ_y, respectively. The kinematics of this design are expressed below, where d represents the direction of the gimbal in rocket coordinates.

$$d = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_y) & -\sin(\theta_y) \\ 0 & \sin(\theta_y) & \cos(\theta_y) \end{bmatrix} \begin{bmatrix} \cos(\theta_x) & 0 & \sin(\theta_x) \\ 0 & 1 & 0 \\ -\sin(\theta_x) & 0 & \cos(\theta_x) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

$$d = \begin{bmatrix} -\sin(\theta_x) \\ \sin(\theta_y)\cos(\theta_x) \\ -\cos(\theta_y)\cos(\theta_x) \end{bmatrix} \tag{4}$$

Since we know the gimbal direction is a unit vector, the third component can be written as a function of the first two components.

$$d = \begin{bmatrix} -\sin(\theta_x) \\ \sin(\theta_y)\cos(\theta_x) \\ -\sqrt{1-\sin^2(\theta_x)-\sin^2(\theta_y)\cos^2(\theta_x)} \end{bmatrix} \tag{5}$$

In global coordinates, the force due to the gimbal is expressed as

$$F^g = -R^r dm(t) u_1 \tag{6}$$

and the force due to the lateral thruster is shown below.

$$F^l = R^r \begin{bmatrix} u_4 \\ u_5 \\ 0 \end{bmatrix} \tag{7}$$

The lateral thrusters generate no torque on the rocket since they are positioned at the center of mass. The torque can be written as the cross product between l and F^g.

$$T = l \times F^g \tag{8}$$

The total sum of forces and torques on the rocket are expressed below as a function of the gimbal direction, the main thrust, and the unknown forces.

$$\begin{bmatrix} F_x \\ F_y \\ F_z \\ T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} F^g + F^l + g \\ l \times F^g \end{bmatrix} + f(x, w) \tag{9}$$

The cross product is a linear operator of the following form.

$$l\times = \begin{bmatrix} 0 & -l_3 & l_2 \\ l_3 & 0 & -l_1 \\ -l_2 & l_1 & 0 \end{bmatrix} \tag{10}$$

This highlights the fact that the control of torque is linearly dependent on the control of the force for a given rocket orientation regardless of control input u.

$$\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} 0 & -l_3 & l_2 \\ l_3 & 0 & -l_1 \\ -l_2 & l_1 & 0 \end{bmatrix} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \tag{11}$$

This result emphasizes the difficulty of this problem. At most, three out of the five derivative state variables can be independently manipulated.

MDP (Paul)

An MDP is used to model decision making in discrete, stochastic, sequential environments. Under the framework, an agent observes a state $s_t \in S$ at each time step t. The agent must then take an action $a \in A$, while trying to maximize the total expected future reward. This process repeats from $t = 0$ to $t = Term$, where $Term$ is the terminal time. The state trajectory is denoted as an episode. For our problem, we are concerned with finite horizon MDPs, which often include a terminal reward function $R^t(S)$. This reward is received only once, when the episode is terminated. The solution to an MDP is a policy π that maps states to actions, which maximize the total expected reward. Every policy has a concept of a value function $V(S)$ as seen in equation (12). The value function indicates the total expected future reward from a given state if the policy π is followed.

$$V^*(s) = \max_a \sum_{s'} P(s'|a,s)\left[R(s,a,s') + \gamma V(s')\right] \tag{12}$$

The optimal policy can be found by solving for the optimal value function $V^*(S)$ with value iteration.

Q-learning is similar to the standard MDP formulation, except that the transition probabilities P are not known. This is known as model free and is advantageous since an explicit model of the process is not required. In this setting, an action value function is used to learn the expected reward of taking any action from any state. The disadvantage of Q-learning is that the number state action tuples grows exponentially with the number of states and actions.

Coupled MDPs (Francesco)

One general method for tackling large MDPs is decomposition (C. Boutilier and Geib 1997). This can be applied when an MDP is specified by a set of "pseudo-independent" subprocesses (Singh and Cohn 1997). Tightly-coupled MDPs are a set of MDPs where the action of one MDP influences the state and available actions of other MDPs. Traditionally, this solution framework is effective when there is a 'main' controlling action that directly influences other available actions. For our problem setup, the 'main' control action is the selection of the thruster power - a construct parallel to the available power of the rocket in a given state.

Symmetrical properties in MDPs (Francesco)

The concept of exploiting symmetry in MDPs has been analyzed in (Zinkevich and Balch 2001) and (Narayanamurthy and Ravindran 2007). The results of these papers confirm that policies can be formulated as functionally homogeneous. If two MDPs are an equivalent homogeneity, then there exists an optimal policy which is symmetric with respect to their equivalence homogeneity. Additionally, these approaches are expected to speed up the convergence to the optimal value function, because the agent learns the strategy for a "smaller" MDP. In the rocket landing problem, we exploit the symmetry of the system, namely in controlling the gimbal and the lateral thrusters.

Hierarchical MDPs (Paul)

The motivation for a hierarchical MDP is mitigating the curse of dimensionality by solving the problem in an abstract space (Mausam and Kolobov 2012). Since memory and computational requirements grow exponentially with the number of states and actions, reducing the state space through a hierarchical structure can lead to tractability. The concept of options is a poplar hierarchical scheme proposed by Sutton (Sutton, Precup, and Singh 1999). In this setting, an MDP is allowed to chose an option MDP as one of its actions. The chosen MDP executes until a termination criteria is met, upon which control is returned to the original MDP. Since the selected MDP executes for a variable amount of time, this results in a semi-MDP.

In hierarchical MDPs, the structure plays a critical role in their performance. Poorly chosen option MDPs will not be beneficial. Some methods have been proposed to automatically build a hierarchical MDP, such as MAXQ (Dietterich 2000). However, the policy learned from MAXQ is not globally optimal, rather it is recursively optimal. Automatic generation of hierarchical MDPs is a difficult task and an ongoing area of research. For our approach, we will inject domain knowledge into the structure by hand crafting relevant sub-MDPs.

RRT - a search approach baseline (Francesco)

Rapidly-exploring random trees (RRT) have shown great success in solving non-convex high-dimensional spaces by building a space-filling tree and were first proposed by (Lavalle 1998). Some high dimensional planing domains have seen great success with RRT, such as robotic manipulation (Dong-Hyung Kim et al. 2013). In particular, an RRT-based implementation enabled a dual arm robot with 14 degrees of freedom to perform assembly and disassembly tasks in a industrial scenario. This exemplifies the effectiveness of RRT-based search for high dimensional continuous problems. We will use RRT as a baseline search approach for our problem.

Formal Problem (Francesco)

We formulate vertical rocket landing with several different MDPs, including a hierarchical MDP structure, given the controls and sensor configuration from Tab. 1 and Tab. 2. The final policy must reach any of the goal states defined in Tab. 3 from any initial state defined in Tab. 4 with at least a 95 percent success rate. We define success as controlling the rocket by using any sequence of available actions which lead the rocket from any initial state to any of the goal states within a 7 second time limit.

control	min	max	units
thrust	0.0	200	N
gimbal angles (X, Y)	-5	5	$^\circ$
lateral thrust (X, Y)	-20	20	N

Table 1: controls

sensor	precision	units
x, y, z	0.5	m
$\dot{x}, \dot{y}, \dot{z}$	1	$^\circ$
ω_x, ω_y	2	$^\circ$/s

Table 2: sensor precision (inertial measurement unit)

Given the continuous nature of this problem, we confine the state of the rocket to be within the ranges specified in Tab. 5. Additionally, the actions of the rocket where selected from the ranges specified in Tab. 6. Perhaps, violating any of the state boundaries would be sufficient to terminate the simulation and invoke a strong penalty to the MDP. We opted instead to let the MDP explore states outside of these boundaries until the simulation time reached 7 seconds. At this point, if the rocket was over 0.5 m in z, we forcefully terminated the simulation with a large penalty. The 7 second simulation boundary was chosen because at this point the rocket's thruster burns out, causing it to lose control.

The OpenRocket simulator (Francesco)

In order to ensure accurate simulations, we opted for the most realistic open-source rocket simulator available today. Our choice was OpenRocket, a model-rocket simulator actively used thousands of members in the rocketry community. A model-rocket simulator is precise enough for our problem because we are operating at low altitude, where wind and other variables have low variation and are easier to model.

Positive aspects (Francesco) This application has over 273 stars on GitHub and over 30 contributors, increasing our confidence on its reliability. Its simulator leverages the 4th order method of Runge–Kutta (RK4), which uses temporal discretization to approximate numerical solutions of ordinary differential equations (Runge 1895). Additionally, OpenRocket supports a framework with simulation listeners,

state	min	max	units
z	0.0	0.75	m
γ_x, γ_y	-4	4	$^\circ$
$\dot{z}$	0.0	0.75	m/s
$\dot{x}, \dot{y}$	-0.75	-0.75	m/s

Table 3: goal states

state	min	max	units
z	28	32	m
γ_x, γ_y	-20	20	$^\circ$
ω_x, ω_y	-26	26	$^\circ$/s
$\dot{z}$	-10	-8	m/s
$\dot{x}, \dot{y}$	-6	6	m/s

Table 4: varying initial states

state	min	max	units
x, y	-20	20	m
z	0	32	m
$\dot{x}, \dot{y}$	-20	20	m/s
$\dot{z}$	-10	2	m/s
γ_x, γ_y	-45	45	$^\circ$
ω_x, ω_y	-45	45	$^\circ$/s

Table 5: state bounds

drastically reducing changes required to the source code of the simulator. The application is implemented to be multi-processed and multi-threaded, by running multiple simulation workers and drawing CPU time from a pool of available threads. We implemented thread-safe code to allow for parallelizable training sessions and take advantage of this implementation.

Negative Aspects (Francesco) OpenRocket doesn't support thrust vectoring - directing the thrust with a gimbal - and did not support forces from lateral thrusters - perpendicular to the rocket, located at its center of mass. Hence, we implemented the dynamics discussed in equation (9).

A dynamic MDP definition framework (Francesco)

In this section, we discuss the criteria required for a dynamic MDP definition framework. We start by analyzing the critical requirements of such a framework, discuss our implementation, and explain our reasoning for our approach.

Default State and Action fields (Francesco)

OpenRocket steps through a simulation by updating a SimulationStatus object. It contains many fields related to the rocket's state, a subset of which are sufficient to define our problem. The inherited **state** fields from the SimulationStatus follow the format "fieldX", "fieldY" and "fieldZ". These state fields are positionX, velocityX, angleX, angleVelocityX, positionY, etc. The **action** fields are: thrust, gimbalX, gimbalY, lateralThrustX, lateralThrustY.

control	min	max	units
thrust	20	180	N
gimbal angles (X, Y)	-3	3	$^\circ$
lateral thrust (X, Y)	-18	18	N

Table 6: action bounds

Framework implementation requirements (Francesco)

A dynamic and flexible MDP definition framework requires the following criteria, supporting:

- flexible MDP definition creation in a standardized format
- MDP definitions in a human-readable and editable format
- the description of state and action variables
- a save-able definition file format
- arbitrary expression definitions, for reward specification and custom fields (e.g. log scale)
- parameter specification for different RL methods (discount, learning rate and exploration rate)
- single MDP implementations
- coupled MDP implementations
- axial-symmetry of specific fields, to enable the re-use a policy around a axially-symmetric transformation
- hierarchical rule-based MDP selection
- hierarchical MDP structure where an MDP selects another MDP as an action

The following criteria are critical, but can rely on additional software:

- plots for definition boundaries and discretizations
- 3D visualization for verification of results

Description implementation and specification

The following paragraphs define the specification format, of which a full example of the standardized format is shown in Fig. 3.

Core definition (Francesco) User-specified MDP definitions are implemented as a Java class, which can be entirely populated by a JSON string. By selecting JSON as our specification format, we are able to allow for MDP definitions to be directly edited and modified without the aid of additional software. An MDP definition has the following required fields: "name", "methodName", "discount", "reward", "stateDefinition" and "actionDefinition". The definition also relies on the fields: "exploration", "alpha" (learning rate), "stepDiscount", "terminalReward" and finally "successConditions" which follows the format "successStateField": [min, max], inclusive.

Custom expressions (Francesco) User-defined expressions were implemented with recursive-descent parsing, and can be evaluated on a state or action, allowing for a complete customization of the reward and terminal reward functions. Additionally, the values of state variables can be assigned directly from custom expressions, enabling definitions to use non-linear scales. The syntax for mathematical functions must be specified starting with an uppercase character, constants must be entirely uppercase, and finally variables only require the first character to be lowercase. Common mathematical functions were implemented, such as Add, Sum, Sub, Mult and Log. Additionally, basic boolean logic was also implemented (And, Or, Not), where 0 is treated as false and any other value is treated as true.

State and Action descriptions (Francesco) State and action boundaries and discretizations follow the format of [min, max, increment]. By using this format, we are able to force the state values within the boundaries defined by the specification. If a value is smaller than the min, then it is set to the min; the same construct is used for the max. When an MDP is hierarchical and selects children MDPs, it must specify the field "childrenMDPOptions". This contains a map from the available actions to the available MDPs for that action. If an action specifies the selection of an MDP, the action field name must contain the string "MDP".

Symmetry definitions (Francesco) The selection of MDPs that use axial symmetry for their variable assignment require the final character of the action field to contain the expected symmetry axis, X or Y. For example, the action controlMDPX selectes an MDP from the "controlMDPX" field of "childrenMDPOptions", and assigns X to the value of that state's symmetry and action's symmetry. The axis string value will be later appended to a given symmetrical field's name for assignment purposes. Axial symmetry of either the X or Y axis can be specified in the "symmetryAxes" field, in which all fields are assigned the value from that component with symmetry defined based on the symmetry of that instance of the state or action. An exemplary use of this field is the use of angle as a state variable, and the adding "angle" in the "symmetryAxes". At runtime, if this MDP is selected for symmetry with the value of the X axis, then the "angle" field will be assigned the value of angleX.

Hierarchical MDP selection expressions (Francesco) By default, an MDP that selects MDPs will be hierarchical and will select an available MDP for a given action. It is also possible to use a rule-base selection, by using the field "MDPSelectionExpressions". This definition field is in the format of a switch-case statement with breaks, in which the first true expression will select that MDP, and the default selection is the last MDP.

3D Visualization and Plotting (Francesco)

Given that we are using OpenRocket as our simulator environment, we opted to integrate visualization and plotting directly within their framework. Notably, our 3D visualization listener works independently of our MDP framework, by obtaining all state variables from the simulation. If a listener extends the "AbstractSimulationListenerSupportsVisualize3DListener", it can define the thrust, gimbal and lateral thrust controls to be used during the visualization. We also extended OpenRocket's built-in plotting capabilities by adding additional state variables to the list of supported plots. In the following paragraphs, we explain how we developed 3D visualization and added custom plotting capabilities to OpenRocket.

3D Visualization (Francesco) We created a simulation listener for 3D visualization, which subscribes to updates of the simulation status at each step of the simulation. Our listener was implemented as an interface, allowing for configurable visualization preferences. For the 3D visualization, we opted for an integration with the open-source Blender application, which supports python scripting. We developed a Python script that runs a TCP server on the local IP address with port 8080, creating flexibility by allowing the visualization to run on a different machine. Fig. 4 contains a screen capture of the 3D visualization with Blender. The ability to visualize the behavior of an MDP real-time is critically important to verify the correctness of the solution. Additionally, this tool is helpful to analyze unexpected behaviors in an MDP's policy.

```
{
  "name": "MDPDefinitionFormat",
  "methodName": "TD0",
  "priority": 1,
  "reward": "Add(-Pow(Todeg(log8Angle),2.0),0.3)",
  "terminalReward": "0",
  "discount": 0.999,
  "stepDiscount": 0.9,
  "alpha": 0.01,
  "exploration": 0.01,
  "symmetryAxes": [
    "angle",
    "angleVelocity",
    "velocity"
  ],
  "passDownSymmetryAxis": true,
  "stateDefinition": {
    "log8Angle": [-0.15, 0.15, 0.03],
    "log2Velocity": [-2.5, 2.5, 0.25]
  },
  "actionDefinition": {
    "selectorMDP": [0.0, 1.0, 1.0]
  },
  "childrenMDPOptions": {
    "selectorMDP": [
      "hierarchical_stabilizer",
      "hierarchical_damper"
    ]
  },
  "expressions": {
    "log8Angle": "Mult(Signum(angle),Log8(Add(Abs(angle),1)))",
    "log2PositionZ": "Log2(Add(positionZ,1))",
  },
  "successConditions": {
    "velocity": [-0.75, 0.75],
    "angle": [-0.0698, 0.0698]
  }
}
```

Figure 3: MDP definition format example

Plotting (Francesco) We implemented custom plotting capabilities for variables defined within the state and action of an MDP definition. An important thing to note is that MDP definition variables are forced within their definition

Figure 4: 3D visualization with Blender

boundaries. Notably, if a custom expression is defined for a state or action variable (e.g. log scale), our implementation will attempt to resolve the new variable name and prefer its values to its respective continuous definition. This greatly enhances debugging an MDP implementation by allowing for plotting the true state and action values of the MDP.

MDP Structure (Paul)

This section formulates each MDP used in the final hierarchical structure and defines their state, action, and reward functions. Notably, MDPs will be discussed in reference to x and y axes. In the symmetry section below, we will argue a single policy can be used for both axes. Furthermore, we use logarithmic transformations defined in equations (13) and (14) on several state variables, allowing for more fine-tuned control around critical regions.

$$k^{l2}(v) = sign(v)log_2(|v| + 1) \tag{13}$$

$$k^{l8}(v) = sign(v)log_8(|v| + 1) \tag{14}$$

Coupled structure (Francesco)

Two or more MDPs are coupled when the actions of one MDP influence the state or actions of another MDP. We aim to reduce the state-space of a single monolithic MDP without reducing the number of variables or their discretization. One approach is to separate the task of controlling the thrust from controlling the gimbals into two MDPs. Then, those MDPs can be coupled on the thrust.

- **lander**: responsible for controlling thrust
- **stabilizer**: responsible for directing the thrust

In order to further decrease the state-space, we will assume that the task of vertical angle stabilization is independent of altitude and velocity. This assumption is reasonable only if we ignore the impact of lateral air drag.

In the following subsections, we will formulate MDPs for stabilizing the rocket's orientation, correcting the rocket's lateral velocity, and landing. Since the number of states grows exponentially with the number of dimensions, we will only include relevant states in each MDP formulation and make several assumptions to significantly reduce the size of the state space. Relevant states are those that play a key role in defining a good reward function and influence the dynamics of the MDPs' objectives.

Stabilizer (Paul) The goal of the stabilizer MDP is to correct the rocket's orientation. According to our model, this is achieved by manipulating the torque via controlling the gimbal and the thrust. According to equation (9), the torque is a function of the rocket's orientation and the control inputs u.

We want to decouple the gimbal control inputs to exploit symmetrical properties. Therefore, we define u_2 and u_3 in the following way.

$$\begin{aligned} \sin(u_2) &= -\sin(\theta_x) \\ \sin(u_3) &= \sin(\theta_y)cos(\theta_x) \end{aligned}$$

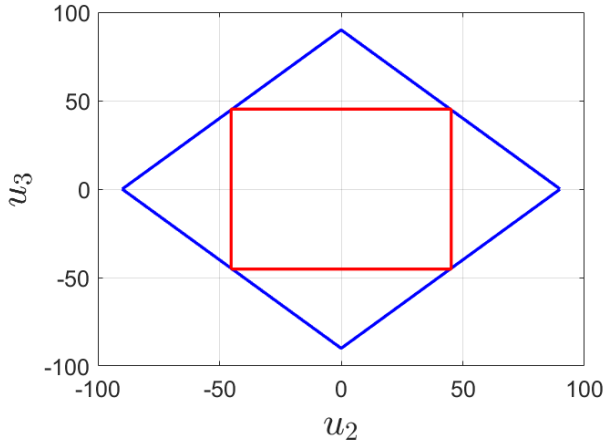

Figure 5: bounds on feasible u_2 and u_3 are shown in blue and the 45 degree limit is shown in red

We can then express the gimbal angles in terms of the control inputs u_2 and u_3.

$$\theta_x = -u_2 \tag{15}$$

$$\theta_y = \sin^{-1}\left(\frac{\sin(u_3)}{\cos(-u_2)}\right) \tag{16}$$

Finally, the gimbal direction d is redefined in the following way.

$$d = \begin{bmatrix} \sin(u_2) \\ \sin(u_3) \\ -\sqrt{1 - \sin^2(u_2) - \sin^2(u_3)} \end{bmatrix} \tag{17}$$

As long as u_2 and u_3 are less then 45 degrees, then there exists a feasible configuration for the gimbal expressed in terms of u. This is illustrated in Fig. 5. This result is important because it allows for the MDP to chose u_2 and u_3 independently, hence, the controls are decoupled.

We want to develop a single controller that corrects the rocket's orientation by manipulating the torques T_x and T_y independently. These torques, however, are functions of both u_2 and u_3. With an approximation, we can treat T_x and T_y as functions of u_3 and u_2 only. If we assume all sin^2 terms are zero, which is a reasonable approximation when both the rocket angles and gimbal angles are close to zero. In our problem, small angles occur during normal operating conditions, corresponding to the rocket being close to vertical. Fig. 6 shows the angle difference in the true gimbal direction and the approximation over a range of rocket angles. Notably, the error introduced by the $\sin^2$ assumption is very small when the rocket is near vertical. As it flips past 45°in both rocket angles, the gimbal angle error can be more than 1°. Using results from equations (9), (20), and (22) and ignoring the unknown dynamics $f(x, w)$, T_x is a function of only u_3 and u_1 and T_y is a function of only u_2 and u_1. The approximate the gimbal forces are as follows.

$$F_x = -(\sin(u_2)\cos(\gamma_x) - \sin(\gamma_x))u_1 m(t) \tag{18}$$

$$F_y = -(\sin(u_3)\cos(\gamma_y) - \sin(\gamma_y)\cos(\gamma_x))u_1 m(t) \tag{19}$$

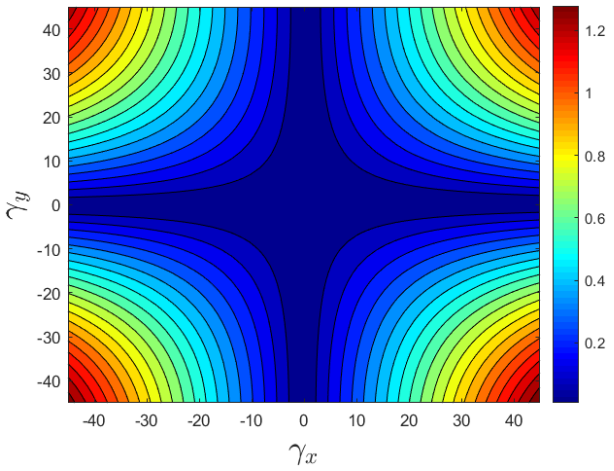

Figure 6: gimbal angle error at different rocket angles γ_x, γ_y due to the $\sin^2$ approximation at $\theta_x = 3°$ and $\theta_y = 3°$

$$F_z = (\cos(\gamma_y)\cos(\gamma_x))u_1 m(t) \tag{20}$$

For any given thrust u_1, the toques T_y and T_x can be independently manipulated by the approximate gimbal forces F_x and F_y.

The rocket axial direction D can be expressed by rotating the $[0, 0, 1]^T$ vector into the global coordinate frame with R^r. The result is shown below.

$$D = \begin{bmatrix} \sin(\gamma_x) \\ -\sin(\gamma_y)\cos(\gamma_x) \\ \cos(\gamma_y)\cos(\gamma_x) \end{bmatrix}$$

The objective of the controller is to select torque values that minimize D_x and D_y and the rocket's angular velocity ω.

Given these results, we define the state for the x and y axes of the stabilizer MDP as the following.

$$S^x = [u_1, k^{L8}(\sin^{-1}(D_x)), \omega_y]^T$$

$$S^y = [u_1, k^{L8}(\sin^{-1}(D_y)), \omega_x]^T$$

The state variables in the above definition are needed to control the dynamics of the rocket's orientation. The actions of the stabilizer MDP is defined as the following.

$$a_x = u_3$$

$$a_y = u_2$$

Finally, the reward needs to reflect the stabilizer's objective to learn a good policy. We want to minimize the rocket's non-vertical axial components, hence, we use the following reward function.

$$R(S) = 0.3 - (S_2 \frac{\pi}{180})^2$$

In summary, the stabilizer (X) controls u_3 with X-axis variables and the stabilizer (Y) controls u_2 with Y-axis variables.

Damper (Paul) In order to reduce the lateral velocity to within the goal bounds, the damper must select appropriate lateral thruster forces u_4 and u_5. Notably, the damper MDP can independently control the x and y axis forces with the same assumption as the stabilizer: for small angles γ_x and γ_y, the lateral thruster in the x axis mainly affects $\dot{x}$ and the lateral thruster in the y axis mainly affect $\dot{y}$. Therefore, the forces in global coordinates are written as follows.

$$F_x = u_4 \cos(\gamma_x) \tag{21}$$

$$F_y = u_5 \cos(\gamma_y) \tag{22}$$

Additionally, the lateral thruster forces also influence the vertical velocity depending on the rocket's orientation. The impact on vertical velocity is negligible since the lateral thrust force is an order of magnitude smaller than the main thrust force. Since $\dot{x}$ and $\dot{y}$ can be directly manipulated by the damper's action, there is no need to include states related to the rocket's dynamics in the state definition. The only states needed are the lateral velocities, which are required to craft a good reward function. The state of the damper is defined as follows.

$$S^x = [k^{L2}(\dot{x})]^T$$

$$S^y = [k^{L2}(\dot{y})]^T$$

The actions of the damper MDP are shown below.

$$a_x = u_4$$

$$a_y = u_5$$

Lastly, the reward function for the damper must penalize high lateral velocities. We define the reward function in equation (23).

$$R(S) = 0.2 - |S_1| \tag{23}$$

In summary, the damper (X) controls u_4 with X-axis variables and the damper (Y) controls u_5 with Y-axis variables.

Lander (Paul) The goal of the lander MDP is to slow the rocket's vertical decent by selecting the thrust. According to equation (9), the vertical acceleration is dependent on the gimbal actions u_2, u_3, and the main thrust u_1. To reduce the size of the state space, we will couple the lander with the stabilizer MDP. If the stabilizer selects the gimbal angles, then the lander can only influence its vertical velocity with the thrust u_1 according to the following equation.

$$F_z = h(S^r)m(t)u(t) + f_z(w)$$

The states that play a key role in the dynamics are time and the actions of the stabilizer and damper. To reduce the state space, we will not include their actions in the lander's state definition. The states needed to formulate a good reward function are height z and vertical velocity $\dot{z}$ because a successful controller will reach the ground with velocity near zero. Therefore, we define the following state for the lander MDP.

$$S = [k^{L2}(z), k^{L2}(\dot{z}), t]$$

Since the lander only has control over the main thruster, we define its action as the follows.

$$a = u_1$$

Finally, we need to design a good reward function for the lander. We will penalize the rocket's vertical velocity on impact with the ground according to the following terminal reward function.

$$R^t(S) = 10000(-|S_2| + 0.1) \tag{24}$$

Additionally, in order to have more time to recover from perturbations, the rocket should reach the ground as soon as possible. Thus, at each time step, we penalize the rocket based on time according to the following reward.

$$R(S) = -t \tag{25}$$

In summary, the lander controls the thrust.

Axial-symmetry (Francesco)

The advantage of exploiting symmetry to reduce the state-space of MDPs has been presented by (Zinkevich and Balch 2001) and (Narayanamurthy and Ravindran 2007). An explanation on how this applies to our problem is contained in the stabilizer and damper sections. The shared thrust among the x and y axes is asymmetric. However, since the stabilizer does not select the thrust, its policy is axial-independent, e.g applying a gimbal force in x manipulates orientation in x in the same way applying a gimbal force in y manipulates the orientation in y. For these same reasons, the damper also axially-symmetric. The damper's controls are subject to the same error found in the stabilizer's controls: if the rocket it tilted, the lateral thrust force in x can influence the velocity in y, and vice-versa. Given this formulation, we can now expand our coupled structure to the following MDPs:

- lander: controls the thrust
- stabilizer (X): controls u_2 with X-axis variables
- stabilizer (Y): controls u_3 with Y-axis variables
- damper (X): controls u_4 with X-axis variables
- damper (Y): controls u_5 with Y-axis variables

Here, stabilizer and damper are a single policy, where X and Y denote the axis on which the policy is applied on.

Hierarchical structure (Francesco)

An MDP structure is considered hierarchical if high level MDPs are choosing among low level MDPs. We take inspiration from the options method, but with a modification. Rather than running the selected MDP until it terminates, we allow the high level MDP to choose between children MDP's at each time instant. We will formulate a high level selector MDP that chooses MDPs as its action, namely the stabilizer and damper.

Selector (Francesco) We implement a selector MDP that chooses between the the stabilizer and damper in order to limit our MDP state-space size. The selector MDP is responsible for minimizing the rocket's orientation and lateral velocity. We must stress advantage of having an additional MDP compared to adding the 'selecting' action directly to the lander, as it would require additional state variables. Notably, the lander and selector are independent, meaning they have peer relationship.

The effect of the selector's action should be influenced by the state of the stabilizer and damper. Ideally, the state definition for the selector would simply be the concatenation of the two state definitions. However, since we are trying to reduce the size of the state space, we cannot include all of them. Instead, we will define the selector's state solely on the state needed to define the reward function. The problem's goal state requires that the rocket angle be vertical and the lateral velocity be minimized, so we define the state as follows.

$$S^x = [k^{L8}(\sin^{-1}(D_x)), k^{L2}(\dot{x})]$$
$$S^y = [k^{L8}(\sin^{-1}(D_y)), k^{L2}(\dot{y})]$$

Once again, we find ourselves with an MDP containing both the state variables of the x and y axes. Naturally, if the stabilizer's and damper's symmetry are both valid assumptions, then we can also defined the selector as an axially-symmetric MDP. The reward function is described in equation (26).

$$R(S) = ((S_1 \frac{\pi}{180})^2 + 0.3) - 10|S_2| \tag{26}$$

The final hierarchical structure contains the MDPs:

- lander: controls the thrust
- selector (X): selects MDP (X) with X-axis variables
- selector (Y): selects MDP (Y) with Y-axis variables
- stabilizer (X): controls u_2 with X-axis variables
- stabilizer (Y): controls u_3 with Y-axis variables
- damper (X): controls u_4 with X-axis variables
- damper (Y): controls u_5 with Y-axis variables

This hierarchical structure has an exponentially smaller state-space than that of a monolithic MDP, and abides by our control constraints. The action and objective of each MDP is clearly defined in Tab. 7. A graphical representation of our coupled symmetrical hierarchical MDP structure is clearly defined Fig. 7.

Implementation And Results (Francesco)

As specified earlier, the objective involves controlling the rocket from the any initial state to any of the goal states. We present an RRT implementation as a baseline search approach and compare it to our hierarchical MDP structure.

RRT implementation (Francesco/Paul)

As a baseline for this problem, we decided to use RRT to search for a successful state trajectory, while using the same discrete actions as our MDP formulation. This was the most similar search baseline we could build to compare our results with. The initial states of an RRT problem instance are the exact those in the problem statement section. The RRT implementation followed from the original formulation from (Lavalle 1998), and is described in Algorithm 1. In our problem, the SELECT_INPUT method randomly samples 50 actions and forward simulates the current state x_{near} for each of them, returning the action that minimizes the distance between x_{near} and x_{rand}. The most expensive portion of RRT is usually simulating the consequences of a given action, and

method	action	objective
lander	thrust	slow vertical landing (terminal velocity)
selector	MDP - select stabilizer or damper	vertical orientation and low lateral velocity
stabilizer	gimbal	vertical orientation
damper	lateral thrust	low lateral velocity

Table 7: actions and objectives of the different MDPs

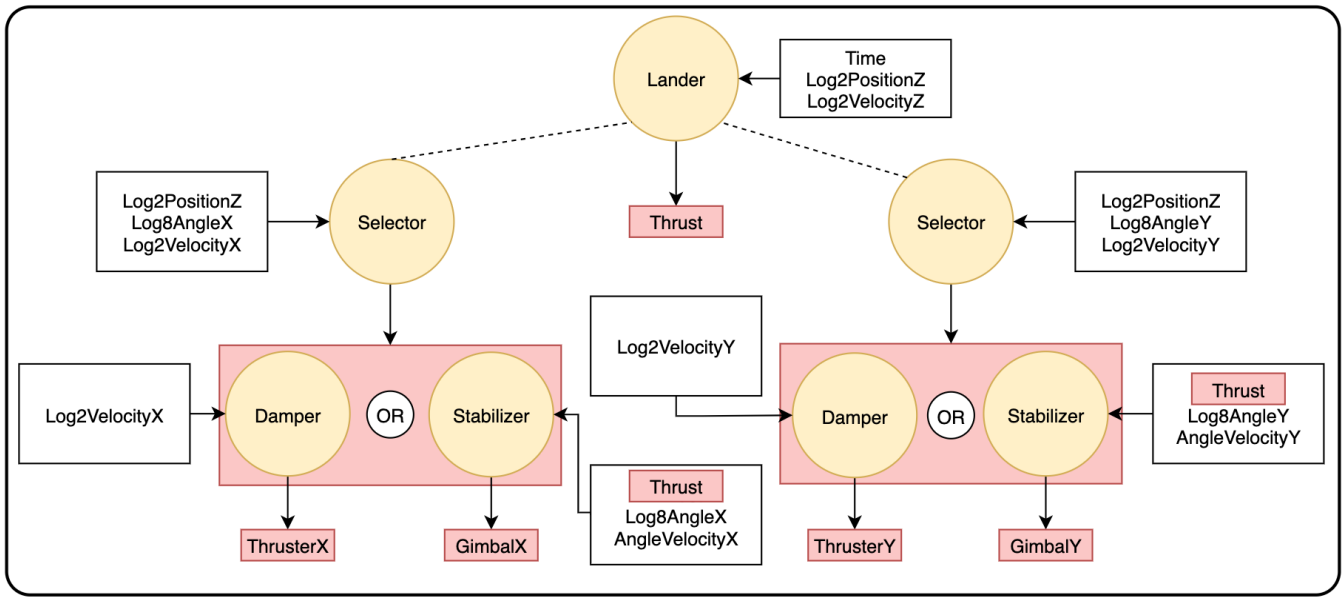

Figure 7: hierarchical MDP structure

Algorithm 1: GENERATE_RRT(x_{init}, K, Δt)

Γ.init($x_i nit$);
for $k = 1$ *to* K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE();
    $x_{near} \leftarrow$ NEAREST_NEIGHBOR(x_{rand}, Γ);
    $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
    $x_{new} \leftarrow$ NEW_STATE(x_{near}, u, Δt));
    Γ.add_vertex(x_{new});
    Γ.add_edge(x_{near}, x_{new}, u);
end
return Γ;

most importantly collision checking. Creating one node, required executing 50 simulation steps, costing approximately 1 ms CPU time. In our problem formulation, no collision checking is required, but each step of the simulator demands drastically more intensive CPU computations compared to finding the nearest neighboring node. For this reason, we did not implement a KD-Tree, which can be used to achieve a rapid nearest-neighbor lookup. A plot showing the state-space explored by all the nodes generated by a single RRT search is shown in Fig. 8. Statistics on the number of nodes required by RRT to find a solution over 50 runs are presented in Fig. 9. Additionally, the minimum number of nodes ex-

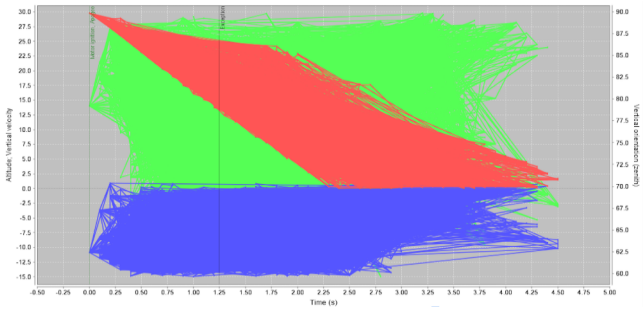

Figure 8: states explored by RRT, where z is red, $\dot{z}$ is blue, and zenith angle is green

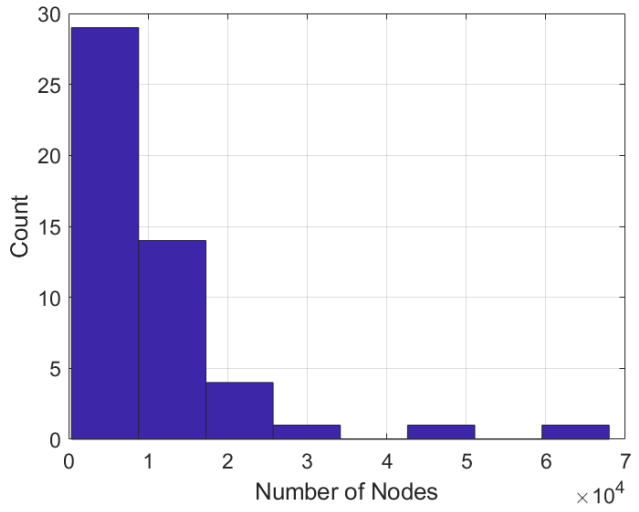

Figure 9: number of nodes generated by RRT over 50 trials

panded is 363, the maximum is 67969, and the median is 7553.

Interestingly enough, the distribution appears to be geometric. A geometric distribution models the probability of an event occurring if there is a probability p for each trial. For RRT, if we think of generating a node as having a constant probability p of finding a solution, then the distribution of the number of nodes needed for RRT to succeed should be geometric. Furthermore, based on distribution in Fig 9, the probability of success for each node expansion is approximately 0.008 percent. With this insight, we triggered restarts for RRT if the number of expanded nodes exceeded 10000, since commonly a solution was found before that threshold.

MDP implementation (Francesco/Paul)

After analyzing the different MDPs, we decided two different RL methods were required, namely Monte Carlo (MC) and Temporal Difference 0 (TD0). The lander's goal is only defined at its terminal state, which is when it hits the ground at the end of the simulation, thus MC is a suitable method. The stabilizer's objective is invariant to termination time as it should always attempt to vertically stabilize the rocket and this same argument still holds for the damper's objective. The selector therefore is also suited to make decisions invariant of termination time, hence we chose TD0 as the RL method for the stabilizer, damper and selector MDPs. For our implementation, we followed the MC and TD0 pseudo-code from (Sutton and Barto 2018) as shown below.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
  $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
  Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

We define the discretization of the MDPs in Tab. 8. We used the same discretizations to create a single monolithic MDP, in which the number state-action combinations was in the order of 10^{13}. Solving this MDP is intractable, and hence we present no results for this approach.

Fig. 10 compares the success rate of our proposed hierarchical MDP structure to the number of training episodes. The hierarchical MDP converged on a policy with 95 percent success rate in just over 10000 training episodes. When this number is compared to the number of nodes expanded in RRT, we see that the required training is quite small. While training the hierarchical MDP, the average episode length is around 100 time steps. The majority of the time, RRT will find a solution after generating 2000 nodes. Generating a node involves simulating a single times step 50 times because of the SELECT_INPUT method. Hence, RRT usually requires simulating 100000 time steps before a solution is found, which is equivalent to 1000 episodes. The main benefit of our approach is that a trained policy can be executed in real time without the need for online planing. Another benefit of this approach is that the final policy is just over 50KB. Therefore, the policy can be run on a microprocessor with very limited memory.

Fig. 11 allows for the analysis of the entire hierarchical system. In the first section of the graph, the selector appears to alternate the selection of the stabilizer and the damper. The second section clearly shows the selector's focus shift to stabilizing the rocket. After the angle is resolved, the selector activates the damper until the lateral velocity is near zero, and in the final section, the stabilizer is left in charge with maintaining the rocket vertical.

Videos are available on YouTube illustrating our results and showcasing the 3D visualisation tool discussed earlier. The hierarchical MDP training is available here: https://youtu.be/Zu5N9lms_EQ. The trained hierarchical MDP is available here:https://youtu.be/r8kyRT0Y35g. Finally, a successful plan found with RRT is available here: https:

	lander	selector (X/Y)	stabilizer (X/Y)	damper (X/Y)
Parameters				
method	MC	TD0	TD0	TD0
alpha	0.01	0.01	0.01	0.01
exploration	0.01	0.01	0.03	0.01
discount	0.999	0.9	0.9	0.9
State				
time	[0, 9, 3]	N/A	N/A	N/A
$k^{L2}(z)$	[0, 5.5, 0.25]	N/A	N/A	N/A
$k^{L2}(\dot{x})$	N/A	[-2.5, 2.5, 0.25]	N/A	[-2.5, 2.5, 0.5]
$k^{L2}(\dot{y})$	N/A	[-2.5, 2.5, 0.25]	N/A	[-2.5, 2.5, 0.5]
$k^{L2}(\dot{z})$	[-5.0, 0.5, 0.25]	N/A	N/A	N/A
$k^{L8}(\sin^{-1}(D_x))$	N/A	[-0.15, 0.15, 0.03]	[-0.05, 0.05, 0.01]	N/A
$k^{L8}(\sin^{-1}(D_y))$	N/A	[-0.15, 0.15, 0.03]	[-0.05, 0.05, 0.01]	N/A
ω_x	N/A	N/A	[-0.21, 0.21, 0.07]	N/A
ω_y	N/A	N/A	[-0.21, 0.21, 0.07]	N/A
thrust	N/A	N/A	[0.1, 0.9, 0.2]	N/A
size	2116	1386	495	11
Action				
throttle	[0.1, 0.9, 0.2]	N/A	N/A	N/A
MDP	N/A	stabilizer | damper	N/A	N/A
gimbal (°)	NA	N/A	[-3, 3, 1]	N/A
lateral thrust (N)	NA	NA	NA	[-18, 18, 3]
size	5	2	7	7

Table 8: implementation details of each MDP

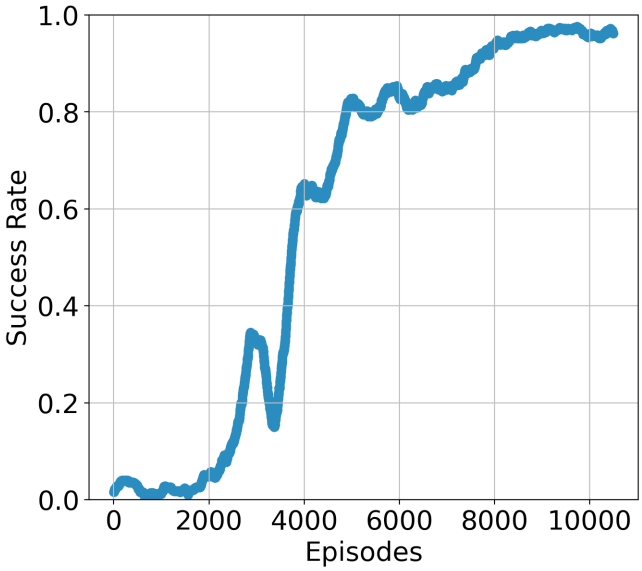

Figure 10: success rate vs. number of episodes for the hierarchical MDP

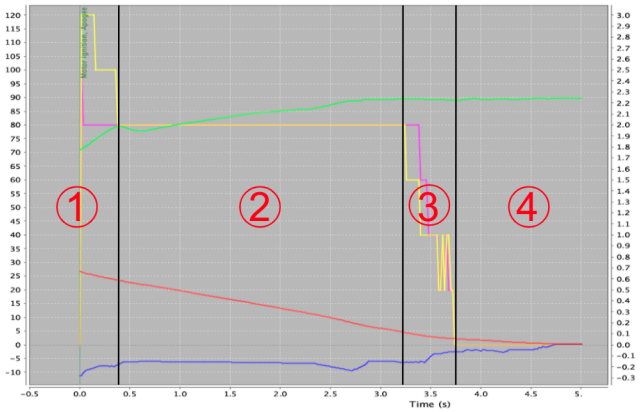

Figure 11: the left plot axis corresponds to z (red), $\dot{z}$ (blue), zenith (green), whereas the right plot axis corresponds to $\dot{x}$ (pink), $\dot{y}$ (yellow)

//youtu.be/kDxorPbG7_I.

Conclusion (Paul)

In this paper, we investigated a hierarchical MDP structure for vertical rocket landing. First, we developed graphical framework for defining MDPs in a standardized format. With this framework, a hierarchical MDP was implemented. We developed several MDP with sub-goals of vertical rocket landing guided by the rocket's dynamic model. Additionally, we exploited coupling and symmetry to make the MDPs computationally tractable. The proposed structure was able to successfully land the rocket within our goal bounds more than 95 percent of the time. Our results indicate that on-line search approaches, such as RRT may not be suitable for our application due to the high computational cost of simulating the rocket's dynamics. In the future, we want open-source our graphical framework and establish vertical rocket landing in a 3D environment as a standard RL benchmark.

References

[C. Boutilier and Geib 1997] C. Boutilier, R. I. B., and Geib, C. 1997. Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. In *IJCAI*, 1156–1163.

[Cohen, Phillips, and Likhachev 2014] Cohen, B.; Phillips, M.; and Likhachev, M. 2014. Planning single-arm manipulations with n-arm robots. In *In Proceedings of the Robotics: Science and Systems Conference (RSS 2014)*.

[Dietterich 2000] Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.* 13(1):227–303.

[Dong-Hyung Kim et al. 2013] Dong-Hyung Kim; Sung-Jin Lim; Duck-Hyun Lee; Ji Yeong Lee; and Chang-Soo Han. 2013. A rrt-based motion planning of dual-arm robot for (dis)assembly tasks. In *IEEE ISR 2013*, 1–6.

[Ferrante 2017] Ferrante, R. 2017. A robust control approach for rocket landing. Master's thesis, University of Edinburgh. https://project-archive.inf.ed.ac.uk/msc/20172139/msc_proj.pdf.

[Goertzel 2009] Goertzel, B. 2009. Cognitive synergy: A universal principle for feasible general intelligence. In *Cognitive synergy: A universal principle for feasible general intelligence*, 464–468.

[Invigorito, Cardillo, and Ranuzzi 2014] Invigorito, M.; Cardillo, D.; and Ranuzzi, G. 2014. Application of openfoam for rocket design. In *OpenFOAM Workshop*.

[Krakauer 2019] Krakauer, J. W. 2019. The intelligent reflex. *Philosophical Psychology* 32(5):822–830.

[Lavalle 1998] Lavalle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning.

[Mausam and Kolobov 2012] Mausam, and Kolobov, A. 2012. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan Claypool Publishers.

[Narayanamurthy and Ravindran 2007] Narayanamurthy, S. M., and Ravindran, B. 2007. Efficiently exploiting symmetries in real time dynamic programming. In *IJCAI*, 2556–2561.

[Runge 1895] Runge, C. 1895. Ueber die numerische auflösung von differentialgleichungen. *Mathematische Annalen* 46:167–178.

[Singh and Cohn 1997] Singh, S. P., and Cohn, D. 1997. How to dynamically merge markov decision processes. In *NIPS*.

[Sutton and Barto 2018] Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

[Sutton et al. 2011] Sutton, J.; McIlwain, D.; Christensen, W.; and Geeves, A. 2011. Applying intelligence to the reflexes: Embodied skills and habits between dreyfus and descartes. *Journal of the British Society for Phenomenology* 42(1):78–103.

[Sutton, Precup, and Singh 1999] Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1):181 – 211.

[Zinkevich and Balch 2001] Zinkevich, M., and Balch, T. R. 2001. Symmetry in markov decision processes and its implications for single agent and multiagent learning. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, 632. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.