Towards Uncertainty-Aware Autonomous Martian Surface Mapping via a Scouting Coaxial Helicopter

Anja Sheppard (anjashep), Jonathan Heidegger (jheidegg), Onur Bagoren (obagoren), and Seth Isaacson (sethgi)
Robotics Department
University of Michigan
Ann Arbor, MI 48109

***Abstract*—This report describes a method for surface mapping of Mars via an autonomous coaxial helicopter. Mars remains a challenging frontier for exploration. Due to the extremely remote nature of the domain, robots on Mars must be autonomous. Autonomous ground vehicles are a common choice, but due to being on the surface, they have limited ability to capture large-scale images. Therefore, in this report, we describe an aerial mapping method for Mars operating on a platform similar to the Ingenuity helicopter. We show how to perform uncertainty-aware exploration of the Martian environment. Further, we show how an aerial vehicle can scout the terrain a ground vehicle will traverse, enhancing the autonomy of the ground vehicle. To accomplish this task, we describe the dynamics and control of a coaxial helicopter. Further, we develop an uncertainty-aware mapping pipeline and a planning and controls system for exploration and scouting. A demo of our system running at 4x speed can be found here.**

I. Introduction

In this project, we describe the nonlinear and linear dynamics of a coaxial helicopter. Coaxial helicopters are a promising design due to their simplicity and few moving parts. For these reasons, NASA selected a model similar to the one we will describe for their Ingenuity project. The helicopter we model consists of two counter-rotating rotors that provide lift and yaw rotation. Additionally, a moving mass actuator sits atop the vehicle's center of gravity and has the effect of moving the overall center of gravity of the system. Notation, coordinate frames, and dynamics are all summarized from [1]. However, [1] provides only the nonlinear dynamics, and hence all linearization work was our own.

Additionally, we implement a cascaded PID positional controller for our vehicle. We demonstrate the capabilities of this controller through a waypoint following task.

Finally, we integrate with perception to perform uncertainty-aware mapping of the Martian surface and a high-level planner that uses uncertainty estimates to plan a route to map a region of the surface.

The rest of the report is organized as follows. In Section II we introduce assumptions, frames, and state definitions. In Section III we describe the nonlinear and linearized vehicle dynamics. In Section IV we introduce a cascaded PID controller for waypoint tracking. In Section V we describe the mapping approach, and in Section VI we describe the high-level planner to perform surface mapping. Finally, Section VII shows experiments and results.

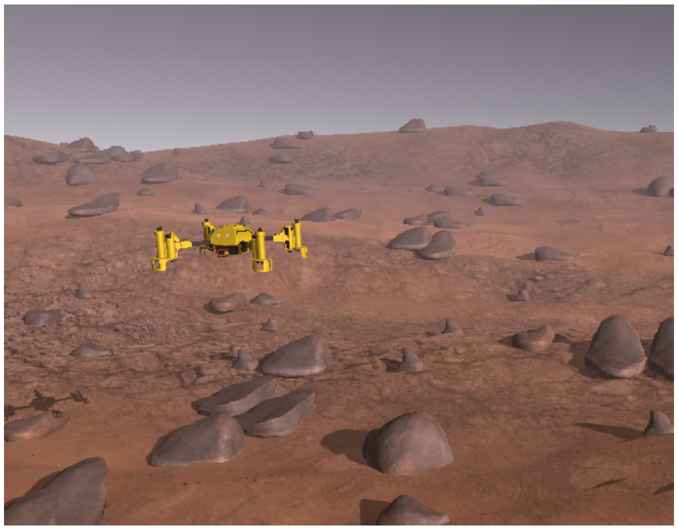

Fig. 1. Drone flying in simulated Martian environment in Unity

II. Definitions

A. Assumptions

We state the following assumptions about our system before we begin defining it further:

1) The mass of the aircraft is constant.
2) Flat Mars is assumed.
3) An Mars-fixed frame $[E]$ is defined whose x and y-axes lie in a horizontal plane and whose z-axis is vertical, with positive axis downward; the Mars-fixed frame is viewed as inertial with origin i.
4) A body frame $[B]$ is defined with origin o located at the center of mass.
5) A wind frame $[W]$ is defined with origin located at the aircraft center of mass, with positive x axis along the velocity vector of the aircraft in the direction of motion, positive z axis normal to the velocity vector and lying in the aircraft plane of mass symmetry, and the positive y axis completing a right hand frame; this wind frame is not a body-fixed frame. We temporarily assume no wind.
6) Gravity is constant and uniform.

7) The atmosphere is stationary with respect to the Mars fixed frame.
8) The aircraft is acted on by aerodynamics, gravitational, and propulsive forces.
9) The propulsion system provides a thrust force on the aircraft that is along the body-fixed z axis of the aircraft.

B. Coordinate Frames and Notation

Before we define the coaxial rotor dynamics, we first present the helicopter model and equations of motion. The inertial frame is centered at the origin i. Our frame of reference is the body frame, with origin o at the drone's center of gravity when hovering, and axes x, y, z. Another frame has origin c at the instantaneous position of the center of gravity of the helicopter.

Mounted atop the center of gravity is a moving mass actuator. A frame centered at the position of the moving mass actuator during hovering has origin s, while a frame with origin at the instantaneous position of the moving mass is at position f.

The center of gravity of the top and bottom rotors are denoted by points t and b respectively, but we assume that the points are nearby and thus only refer to t. Finally, we state that the axis shaft intersects the vehicle body at a joint at position j.

Vectors between points a and b are denoted r_{ab}, with norm d_{ab}. The rotation matrix R_o gives the rotation from the origin to the inertial frame.

Let M be the vehicle's mass, and m the mass of the moving mass actuator.

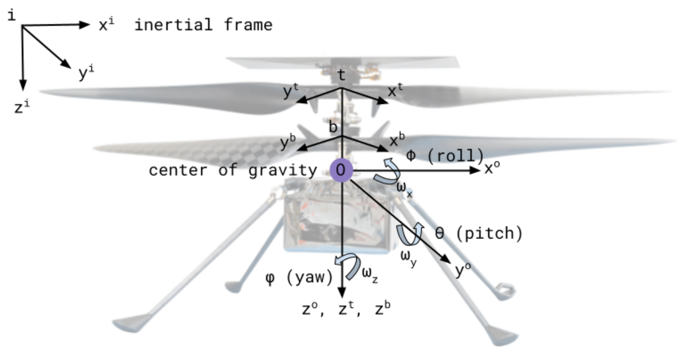

Fig. 2. Coordinate frame definition for our coaxial helicopter model.

C. States and Controls

We denote the rotation by Euler angles $\Theta = [\phi, \theta, \psi]$. Linear velocities are given by $V = [u, v, w]$ and angular velocities are $\omega^o = [p, q, r]$. Other auxiliary variables will be added to the state as needed.

Control inputs are $[\tau_t, \tau_b, F^o_{A_x}, F^o_{A_y}]$ where τ are the top and bottom motor torques and F_A are forces applied to the moving mass actuator. These will be simplified for the linearized model.

III. Vehicle Dynamics

In this section we describe the nonlinear and linearized vehicle dynamics.

A. Rotational Kinematics

We define the rotational kinematics using standard roll, pitch, and yaw (ϕ, θ, ψ) angles:

$$\Theta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{1}$$

The angular velocity (ω) of the helicopter is derived from the derivative of the rotation matrix:

$$\omega^o = \begin{bmatrix} \dot\phi - \dot\phi \sin\phi \\ \dot\theta \cos\phi + \dot\psi \cos\theta \sin\phi \\ -\dot\theta \sin\phi + \dot\psi \cos\theta \cos\phi \end{bmatrix} \tag{2}$$

$$= \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta \sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \dot\Theta \tag{3}$$

Using the previous equation, we can define the rotational kinematics of the coaxial helicopter as the inverse of $Y(\dot\Theta)$:

$$Y^{-1}(\Theta) := \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi \sec\theta & \cos\phi \sec\theta \end{bmatrix} \tag{4}$$

such that

$$\boxed{\dot\omega^o = Y^{-1}\dot\Theta.} \tag{5}$$

B. Nonlinear Dynamics

We first present a nonlinear model with the equations of motion. In the following section we make several additional simplifying assumptions and linearize the dynamics. For a complete derivation, we refer the reader to [1]. Here, we summarize the results.

1) Rotational Dynamics: Define I^o_c as the vehicle's inertia matrix, and define $I_r = \text{diag}(I_{r_{xx}}, I_{r_{xx}}, I_{zz})$ as the inertia of the rotors. Next, define $J^o_r = \frac{2d_{jo}}{d_{jt}}\text{diag}(I_{r_{xx}}, I_{r_{yy}}, 0)$. Let L^o_c be the body-referenced angular momentum of the vehicle at the COG, given by

$$L^o_c = \frac{d_{jo}}{d_{jt}} \begin{bmatrix} 0 & I_{r_{xx}} - r_{yy} & 0 \\ I_{r_{xx}} - I_{r_{zz}} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{6}$$

Finally, define Ω^o_t and Ω^o_b as the rotational velocities of the rotors. These are governed by the following dynamics, where b is a constant:

$$I_{r_{zz}}\dot\Omega^o_t = \tau^o_t - b(\Omega^o_t)^2 \tag{7}$$

$$I_{r_{zz}}\dot\Omega^o_b = -(\tau^o_b - b(\Omega^o_b)^2) \tag{8}$$

$$\tag{9}$$

With these quantities defined, we can assemble the rotational dynamics. We omit the derivation for brevity, but the equations result from four factors: torque due to the rotors pushing on the joint, torque due to the engines, torque induced by the reaction forces of the moving mass actuator, and torque resulting from aerodynamic thrust. The reaction forces from the moving mass actuator are generally minimal, and can thus (according to [1]) can be ignored. With that, the rotational dynamics are as follows:

$$\begin{aligned}(I_c^o + J_r^o) + \dot{\omega}^o + [(\Omega_t^o + \Omega)]\omega^o + \omega^o \times I_c^o \omega^o \\ = \begin{bmatrix} T r_{oc_y}^o \\ -T_{oc_x}^o \\ -\tau_t^o + \tau_b^o \end{bmatrix}\end{aligned} \tag{10}$$

where $T = \Gamma(\Omega_b^2 + \Omega_t^2)$ and Γ is a constant.

2) Translational Dynamics: For the translational dynamics, start with a helper function:

$$\begin{aligned}&\mu(\bar{r}_{oc}^o, \dot{\bar{r}_{oc}^o}, \omega^o, \dot{\omega}^o) = \\ &e_3^T \Bigg(\frac{m}{M}(\dot{\omega}^o \times r_{os}^o + \omega^o \times (\omega^o \times r_{os}^o)) \\ &+ 2\omega^o \times \begin{bmatrix} \dot{\bar{r}_{oc}^o} \\ 0 \end{bmatrix} + \dot{\omega}^o \times \begin{bmatrix} \bar{r}_{oc}^o \\ 0 \end{bmatrix} \\ &+ \omega^o \times \left(\omega^o \times \begin{bmatrix} \dot{r}_{oc}^o \\ 0 \end{bmatrix} \right) \Bigg)\end{aligned} \tag{11}$$

With this in mind, the translational dynamics are given by the following, after assuming that the reaction forces caused by the moving mass actuator are negligible:

$$\begin{aligned}\ddot{r}_{io}^i = R_o \Bigg(&\frac{m}{M}(\dot{\omega}^o \times r_{os}^o + \omega^o \times (\omega^o \times r_{os}^o) \\ &- \begin{bmatrix} 0 \\ 0 \\ \mu(\bar{r}_{oc}^o, \dot{\bar{r}_{oc}^o}, \omega^o, \dot{\omega}^o) \end{bmatrix} \Bigg) + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}\end{aligned} \tag{12}$$

C. System Linearization

In order to linearize the system to bring to a form of

$$\dot{x} = Ax + Bu \tag{13}$$

we initially identify the steady flight conditions. We identify that the steady flight are the conditions where the helicopter is hovering without any position or orientation change, and with zero velocity. We formally define this through setting the values for the state variables under steady flight

$$\Theta = \begin{bmatrix} 0 & 0 & \psi_0 \end{bmatrix} \quad V = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \quad \omega^0 = [0, 0, 0] \tag{14}$$

Hence, we linearized around these steady state values, denoting the delta between the equilibrium and actual values with Δ.

When linearizing the given dynamic models, we make the following assumptions [1]:

1) In steady-state flight and in the operation of the vehicle, the position of the moving mass that enables translation in the xy-plane is close to the center of gravity (COG),
2) The actuator force is negligible during the vehicle operation,
3) The moving mass has a negligible effect on the inertia matrix I_c^o,
4) The gyroscopic effect from the rotation of the rotors are negligible,
5) The helicopter inertia matrix I_c^o is a diagonal matrix, enabling us to ignore the rotor dynamic and its effects on the vehicle dynamics.

Knowing that for the condition for steady flight requires active actuation from the rotors, we keep the rotor angular velocities Ω_t^o, Ω_b^o modeled as free variables of the system, acting as control inputs. Note that due to the above assumptions, we can control directly for the angular velocities rather than motor torques τ. Hence, the control inputs are Ω_t, Ω_b, $r_{oc_x}^o$, and $r_{oc_y}^o$. We linearize these around $\Omega_{t_0}, \Omega_{b_0}, 0$, and 0, respectively using Δ to indicate deviation from their equilibrium values.

We additionally note that for the formulation of our nonlinear dynamics, the acceleration of the system $\ddot{r}_{io}^o$ is tracked in the inertial frame. This results in gravity g being part of the dynamics that are modeled, but added as an additional component to the tracked state variables. In order to account for this, we linearize an augmented state representation, such that we can include the gravity effect in our state transition matrix and include in our stability analysis. Hence, the overall state vector is $[u, v, w, p, q, r, \phi, \theta, \psi, 1]^T$, where 1 assists with the augmented state. Recall that in practice all variables are deviations from the linearization point.

The linearized dynamics of the augmented state vector results in the following state transition matrix. We omit the detailed derivation here, and instead refer readers to the handwritten notes at the end of the document.

$$A = \begin{bmatrix} & \zeta\sin(\psi_o) & -\zeta\cos(\psi_o) & 0 & 0 \\ & \zeta\cos(\psi_o) & -\zeta\sin(\psi_o) & 0 & 0 \\ & 0 & 0 & 0 & g-\zeta \\ & 0 & 0 & 0 & \beta_{1,3}\gamma \\ 0_{10x6} & 0 & 0 & 0 & \beta_{2,3}\gamma \\ & 0 & 0 & 0 & \beta_{3,3}\gamma \\ & 1 & 0 & 0 & 0 \\ & 0 & 1 & 0 & 0 \\ & 0 & 0 & 1 & 0 \\ & 0 & 0 & 0 & 0 \end{bmatrix} \tag{15}$$

where $\zeta = \frac{\Gamma}{M}\left(\Omega_{t_o}^2 + \Omega_{b_o}^2\right)$, $\beta = (I_c^o + J_r^o)^{-1}$, $\gamma = b(-\Omega_{t_o}^2 + \Omega_{b_o}^2)$, and $T_o = \Gamma(\Omega_{t_o}^2 + \Omega_{b_o}^2)$.

Finally, we present the control matrix with the control vector $[\Omega_t, \Omega_b, r_{oc_x}^o, r_{oc_y}^o$. Recall that in practice all variables are deviations from the linearization point.

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{2\Gamma}{M}\Omega_{t_o} & -\frac{2\Gamma}{M}\Omega_{b_o} & 0 & 0 \\ 0 & 0 & -\beta_{1,2}T_o & \beta_{1,1}T_o \\ 0 & 0 & -\beta_{2,2}T_o & \beta_{2,1}T_o \\ 0 & 0 & -\beta_{3,2}T_o & \beta_{3,1}T_o \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{16}$$

D. Stability Analysis

We find that the system is marginally stable, in that the eigenvalues of the system are all zeros. Hence, a controller will be needed to stabilize flight. Additionally, we will consider modeling wind as a disturbance by adding in a wind frame. Finally, Γ is a key parameter for the flight performance, but was not provided in the paper. However, it is controllable through design of the rotors by modifying the rotor geometry and angle of attack. Therefore, we will consider it a free variable in our design and use our results to recommend a value that maximizes the stability of the system.

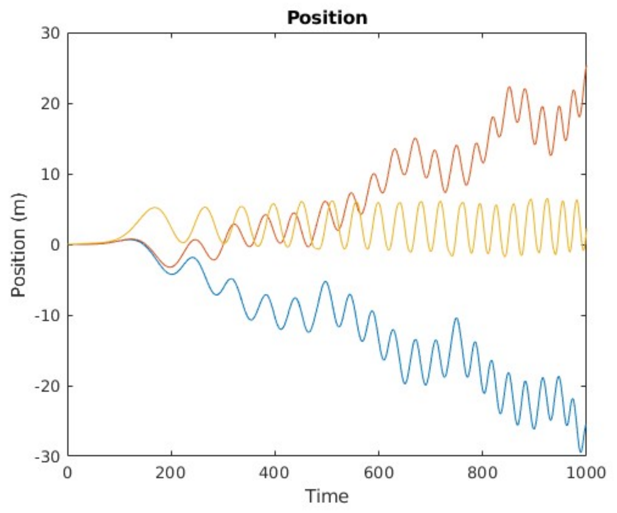

Fig. 3. Simulated trajectory of the system over 10 seconds. Note that the vehicle initially falls as the rotors increase in velocity, then the vehicle flies away and diverges.

E. Simulation

The vehicle's nonlinear dynamics are simulated for 10 seconds. Euler integration is performed using a timestep of 0.01 seconds. The inputs to the system are held constant rotor torque and mass position (0.00305 N-m to each rotor and 0.0005m displacement to each axis of the moving mass actuator). As seen in Fig. 3-5 the vehicle is not stable in this open loop configuration and will need an active control loop for flight.

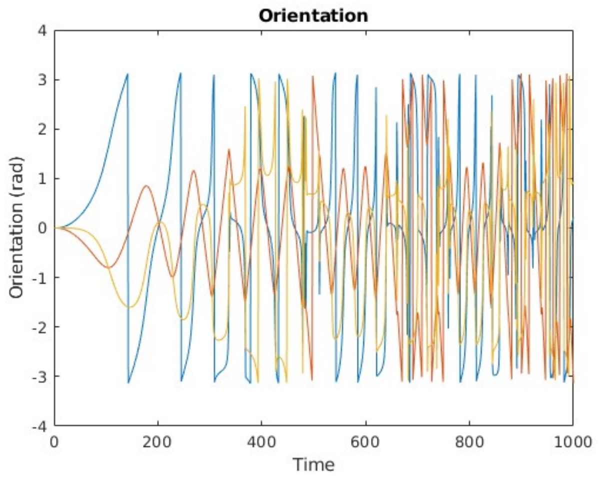

Fig. 4. Simulated orientation of the system over 10 seconds.

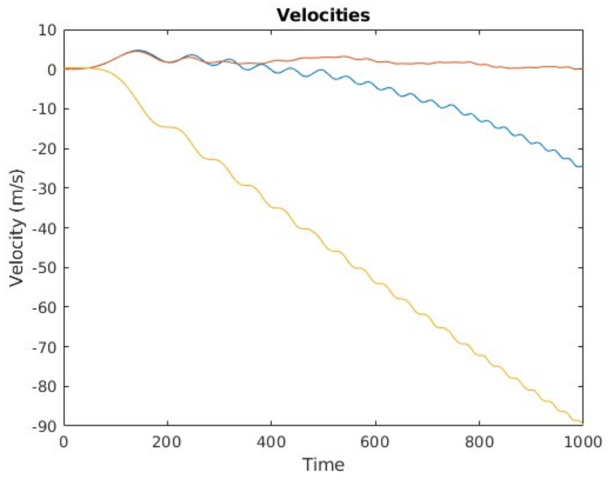

Fig. 5. Simulated velocity of the system over 10 seconds

IV. Vehicle Controller

The nonlinear dynamics of the coaxial helicopter are controlled by a cascaded sequence of PID controllers, some of which include feedforward action. The dynamics decouple well with the control input. The rotor torques are are driven to control the yaw and altitude of the helicopter. The moving mass position drives the translational dynamics in the XY plane and the pitch and roll of the vehicle. The full diagram of the control architecture is given in Fig 6.

A. Rotor Controller

The rotor controller is constructed to achieve setpoints in yaw and rotor speed (which is proportional to the lift generated). First, a PD controller for the yaw commands a torque differential between the two rotors to induce a moment on the vehicle itself. This is added with a rotor speed controller which uses a feed-forward for the rotor speed at steady state

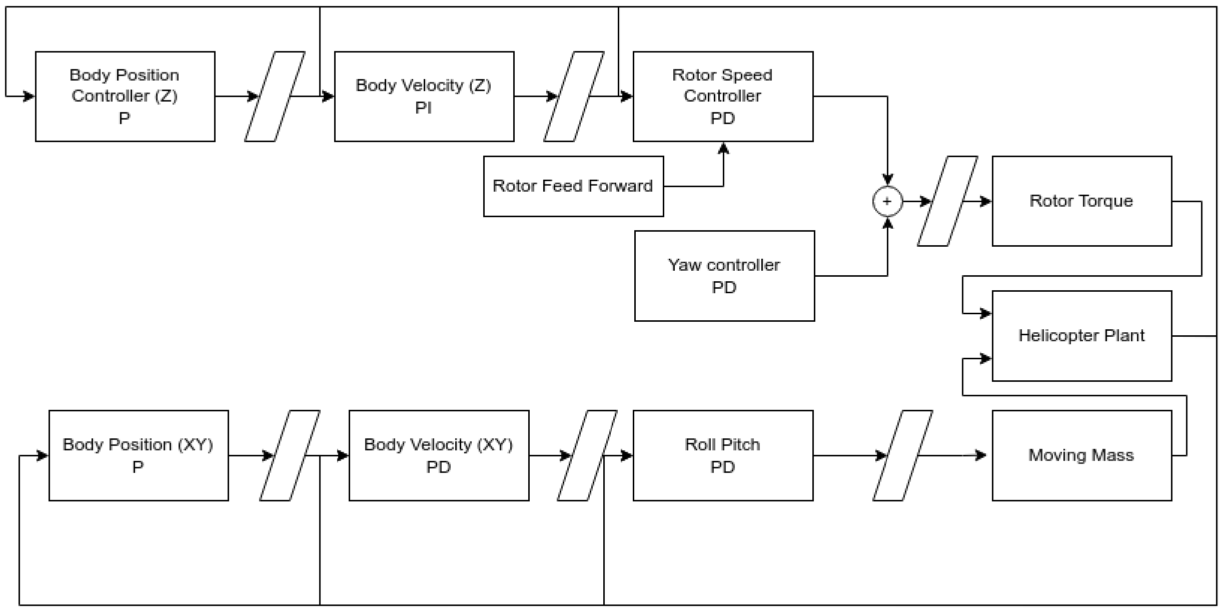

Fig. 6. Flow diagram of controller design with saturation filters between most steps

hover. Between each step saturation filters are used to aid in tuning by keeping the system bounded.

$$\tau_t = -K_{p1}(\psi - \psi_{goal}) - K_{D1}\omega_z + \tau_{rs}$$

$$\tau_b = K_{p1}(\psi - \psi_{goal}) + K_{D1}\omega_z - \tau_{rs}$$

Rotor speed controller torque combines with a feed forward torque needed to maintain hover speed:

$$\tau_{rs} = K_{p2}(\Omega - \Omega_{goal}) + K_{D2}\dot{\Omega} + \tau_{ff}$$

A body frame controller for Z sets the Ω_{goal}:

$$\Omega_{goal} = K_{p3}(V_z - V_{zgoal}) + K_{I3}\int(V_z - V_{zgoal}) + \Omega_{ff}$$

Finally a proportional controller is used to achieve setpoints.

$$V_{zgoal} = K_{p4} * (Z - Z_{goal})$$

B. Moving Mass Controller

The moving mass controller first controls the roll and pitch and by setting setpoints for the orientations V_x and V_y.

$$r_x = K_{p5} * (\theta - \theta_{goal}) + K_{D5}\omega_y$$

$$r_y = -K_{p5} * (\phi - \phi_{goal}) - K_{D5}\omega_x$$

$$\theta_{goal} = K_{p6}(V_y - V_{ygoal})$$

$$\phi_{goal} = -K_{p6}(V_x - V_{xgoal})$$

The position controller then drives the velocity setpoints with a simple proportional controller:

$$V_{xgoal} = K_{p7}(X - X_{goal})$$

$$V_{ygoal} = K_{p7}(Y - Y_{goal})$$

C. Tuning

We took an engineering-based approach to tuning our controller. We manually varied the PD parameters and monitored the response via the simulated position plots. The resulting tuning values are below.

TABLE I
TUNING VALUES FOR THE CONTROLLER

Loop	kP	kI	kD	Ω_{ff}
1	1.0e-3	0	1.0e-3	0
2	3.0e-3	0	8.0e-4	0
3	6.0e-1	1.0e-3	0	2.743
4	3.0	0	0	0
5	5.0e-2	0	5.0e-3	0
6	1.0e-1	0	0	0
7	5.0e-1	0	0	0

V. UNCERTAINTY-AWARE MAPPING

A. Mesh Projection

In order to build a probabilistic elevation map around our robot, we use a Bayesian inference approach to create a 2.5D mesh where the elevation of the mesh at each vertex is treated as a random variable. We then utilize the map elevation uncertainty to guide our planning algorithm. This is based off of the approach in [2], and we use the implementation from [2].

We define a mesh which contains vertices $v = \begin{bmatrix} v_x & v_y & v_z & v_{\sigma^2} \end{bmatrix} \in \mathcal{V}$ and faces $\xi \in \Xi$:

$$\mathcal{V} \subset (\mathbb{R}^4)^m \quad (17)$$

$$\Xi \subset \mathcal{V}^3 \times (\mathbb{R}^3)^l \times \mathbb{R}^k_{\geq 0} \quad (18)$$

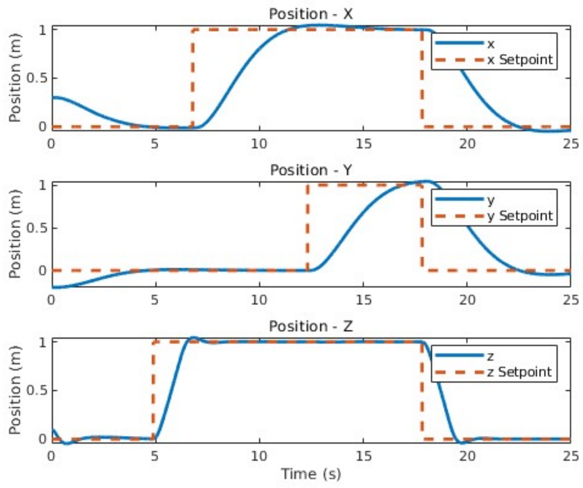

Fig. 7. Position vs. commanded position waypoints

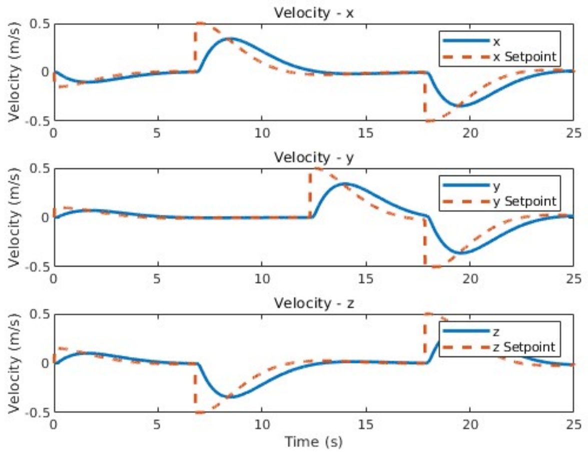

Fig. 9. Velocity vs. commanded velocity waypoints

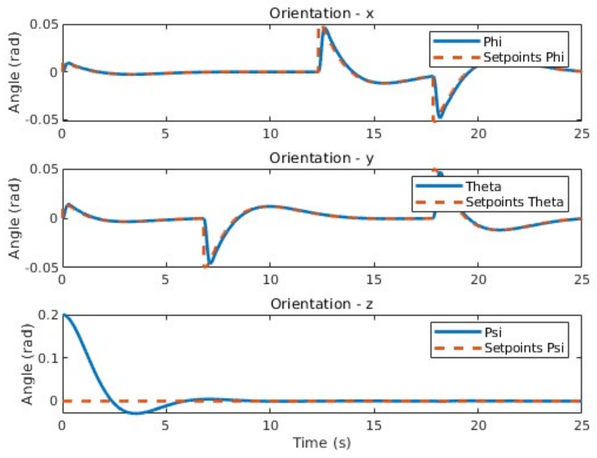

Fig. 8. Orientation vs. commanded orientation waypoints

At each timestep we received co-located RGB and depth images[1] from Unity. The depth image is then formulated as a pointcloud using the camera intrinsics:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{\text{camera}} = D(u,v)K^{-1}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{\text{image}} \tag{19}$$

and then transformed into the world frame using camera extrinsics:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}_{\text{world}} = R_{\text{camera}^{\text{world}}}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{\text{camera}} + t_{\text{camera}}^{\text{world}} \tag{20}$$

[1]https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/depth-map-package-77850

The resulting pointcloud P in the map frame has N points and is thus defined as $P_{\text{map}} \in \mathbb{R}^{3\times N}$. We now project each point in P onto our 2.5D mesh by using Barycentric coordinates, which work well with the 2D simplices making up our mesh. For each point $\boldsymbol{p} \in P$ and vertex $\boldsymbol{v} \in \mathcal{V}$, we take the x and y coordinates to formulate $\tilde{\boldsymbol{p}} = \begin{bmatrix} p_x & p_y \end{bmatrix}$ and $\tilde{\boldsymbol{v}}_i = \begin{bmatrix} v_{x_i} & v_{y_i} \end{bmatrix}$. We then convert $\tilde{\boldsymbol{p}}$ into barycentric coordinates to determine which face of the mesh it is contained by:

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ v_{x_1} & v_{x_2} & v_{x_3} \\ v_{y_1} & v_{y_2} & v_{y_3} \end{bmatrix}^{-1}\begin{bmatrix} 1 \\ p_x \\ p_y \end{bmatrix} \tag{21}$$

If each λ_i satisfies $\lambda_i \in (0,1)$, then we assign that point $\boldsymbol{p}$ to the face ξ made up of vertices $\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}$.

B. Mesh Elevation Update

After we have projected our pointcloud P onto our mesh, we calculate an elevation estimate for each face with a 1D Kalman filter.

For each of the interior points on a given face ξ_i, we calculate the height variance as done by [2], where the Jacobians for the sensor measurement and and sensor frame rotation are derived in [3]:

$$\sigma^2 = J_s \Sigma_s J_s^\top + J_p \Sigma_p J_p^\top \tag{22}$$

Then an elevation estimate v_z and uncertainty v_{σ^2} are estimated via filtering:

$$v_z \leftarrow \frac{v_z \cdot \sigma^2 + z \cdot v_{\sigma^2}}{v_{\sigma^2} + \sigma^2} \tag{23}$$

$$v_{\sigma^2} \leftarrow \frac{v_{\sigma^2} \cdot \sigma^2}{v_{\sigma^2} + \sigma^2} \tag{24}$$

This update happens recursively and yields an updated elevation mesh at each time step. We then use the mesh geometry and the uncertainty at each vertex to plan exploratory trajectories.

In this report, we use the open-source implementation provided by [2].

VI. High-Level Planner

A. Uncertainty-Aware Planning

We utilize the probabilistic height map representation to inform our planner of the next position to go to. We formulate the planning problem as an uncertainty minimization objective, where the uncertainty is from the elevation map vertices $\mathcal{V}$.

We specifically take an approach that maximizes coverage while reducing uncertainty. The objective is to observe the vertex $v^* \in \mathcal{V}$ with the highest uncertainty that is a threshold distance $d_{\text{threshold}}$ away from the current pose of the robot $p \in \mathbb{R}^3$. This objective is shown in eq. (25).

$$\begin{aligned} v^* &= \underset{v \in \mathcal{V}}{\text{argmax}}\ v_\sigma^2 \\ \text{s.t. } & \|p - v\|_2 \geq d_{\text{threshold}} \end{aligned} \tag{25}$$

In order to ensure that we can observe v^*, we plan a path to $(v_x^*, v_y^*, v_z^* + 10)$ so that we do not collide with the terrain when navigating.

B. Trajectory Tracking

We additionally designed a planner to act as a surveyor for an unmanned ground vehicle (UGV) traversing across the surface of Mars during a scientific mission. In order to make sure that the path the UGV is traversing is safe, it is useful to have a map if it beforehand. To enable this, we use our coaxial helicopter to generate an elevation map while traversing through the path that the UGV is planned to drive through.

In order to enable this, we take key waypoints along the path that an UGV will follow and perform a cubic interpolation on the $x - y$ points to generate a continuous function that fits through the waypoints. We set the height of the robot to be conservatively above the terrain and used this as our path to survey the area.

VII. Experiments

The described controller, mapper, and planner were implemented in a simulation environment. We conducted experiments to verify both planning modes.

A. System Architecture

The system architecture for our experiments is shown in Figure 10. A custom simulator combining MATLAB dynamics and Unity graphics provides RGB and depth images to the SEL MAP mapping package, which implements the mapping described in Section V, as implemented by the authors of [2]. SEL MAP sends uncertain-aware depth maps to the planning module, which either tracks a target trajectory or performs uncertainty-aware exploration, as described in Section VI. Finally, a PID controller is implemented in MATLAB as described in Section IV. All components coordinate with eachother over ROS.

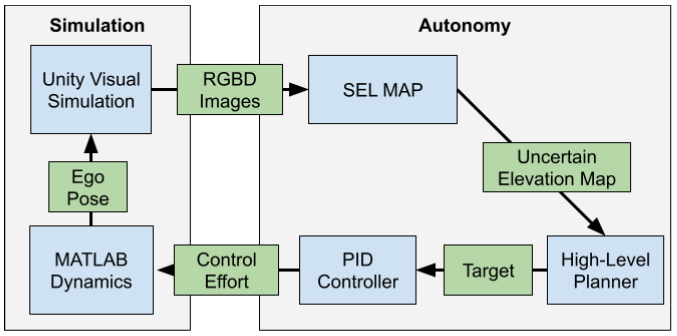

Fig. 10. A Unity and MATLAB simulation provides sensor measurements to the autonomy stack, which contains perception, planning, and control modules.

B. Simulation

To simulate the vehicle dynamics, the nonlinear equations of motion as described in Section III were implemented in MATLAB. The discrete time non-linear dynamics are controlled by the described cascaded PID controller to track waypoints from the planner. Then the resulting Pose of the helicopter for the timestep is sent to Unity for visualization.

To simulate the visual environment, we used a Unity simulation. The Unity simulation includes an open-source Mars terrain model[2]. The helicopter's dynamics are controlled via MATLAB and published to Unity through a ROS interface. Then, cameras mounted to the simulated robot transmit RGB and depth images, using a combination of open-source libraries[3] and custom ROS extensions to Unity to allow for publishing depth images and camera info messages.

To remain consistent with the limited sensor and compute availability on Mars, we simulate low-resolution 340p images at a frame rate of 0.5hz.

C. Trajectory Tracking

To evaluate the surveying trajectory-tracking, we instruct the robot to follow the trajectory of a UGV. For this experiment, we manually set waypoints for the UGV to follow, then interpolate them with a cubic spline.

1) Results: To ensure accuracy, the vehicle was required to fall within 0.1m of each waypoint before moving on to the next. On average, waypoints were 1.2m apart and the vehicle reached the 0.1m threshold after 6.9seconds. This timing is within reason for Mars vehicles, which tend to operate slowly. The elevation map following the surveying pass is shown in Figure 11.

D. Uncertainty-Based Exploration

To evaluate the ability of our model to reduce the uncertainty in the map, after completing the surveying, we switch the system to an uncertainty-based explorer and allow it to

[2]https://assetstore.unity.com/packages/3d/environments/landscapes/mars-landscape-3d-175814

[3]https://github.com/siemens/ros-sharp

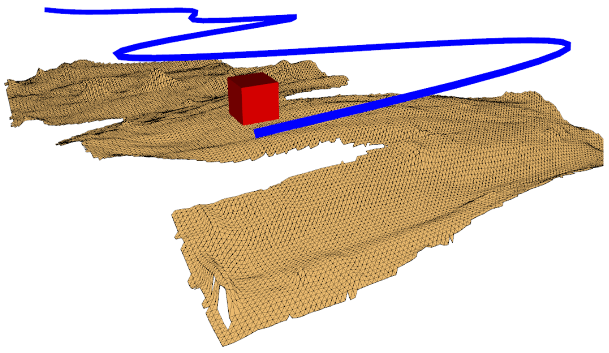

Fig. 11. The vehicle followed the blue trajectory, and produced the elevation map shown in orange.

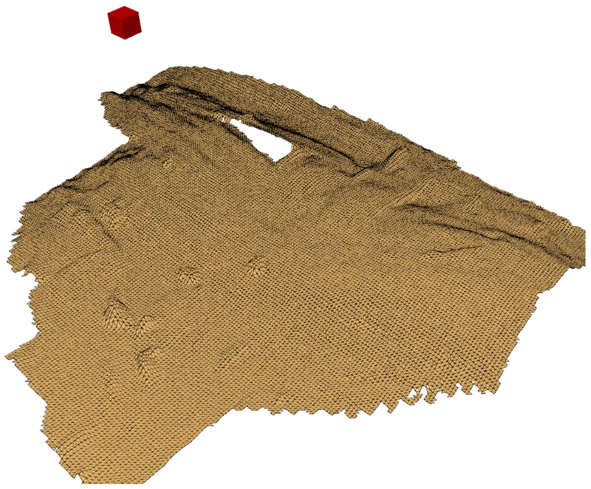

Fig. 12. After running the uncertainty-based exploration, the robot fills in gaps in the map and expands the mapped region.

continue running for 5 minutes. Figure 12 shows how the method filled in gaps from the surveying pass and expand the mapped region.

E. Demo Video

A demo of our system can be found here. This video (sped up 4x) shows the vehicle first tracking a UGV's trajectory for scouting. Then, it switches to the uncertainty-minimizing explorer. In the uncertainty-based exploration mode, a point cloud is used to visualize the uncertainty, where the height and color (red is low and green is high) of each point correlates with the uncertainty.

VIII. Conclusion

In conclusion, we define and derive the nonlinear and linearized model of a coaxial helicopter under the assumptions stated. Additionally, we show the stability of the system using simulations. For our control system, we implemented cascaded a PID controller and demonstrated it with a waypoint following task. We modified our Matlab implementation to connect with ROS and Unity for perception and terrain mapping. Simulated Martian terrain maps were used to demonstrate the exploration and mapping features of the perception and planning systems. In the future, this work could be extended to include wind modeling to test our controller in more hazardous conditions that are present on Mars. Additionally, this work could be expanded upon by including learning-based approaches to safe landing site detection and obstacle identification.

References

[1] F. Zare Seisan, "Modeling and Control of a Co-Axial Helicopter," Master's thesis, University of Toronto, Canada, Jan. 2012.

[2] P. Ewen, A. Li, Y. Chen, S. Hong, and R. Vasudevan, "These maps are made for walking: Real-time terrain property estimation for mobile robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7083–7090, 2022.

[3] P. Fankhauser, M. Bloesch, C. Gehrig, M. Hutter, and R. Siegwart, *ROBOT-CENTRIC ELEVATION MAPPING WITH UNCERTAINTY ESTIMATES*, pp. 433–440.