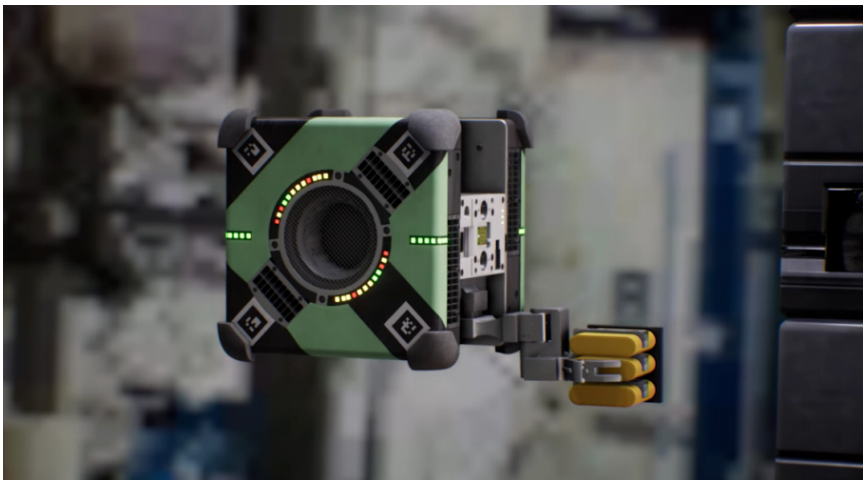
Multivariable Adaptive Robust Control for a Free-Flying Space Robot*

Tammer H. Barkouki, Jason Dekarske, Abhinav G. Kamath



*https://github.com/abhinavkamath/Space-Robot

Contents

Abstract

We demonstrate the ability of a free-flying space robot named Astrobee to follow a moving target using simulated visual navigation. The Astrobee project out of NASA Ames Research Center has built a small fleet of three free-flyers currently on ISS and undergoing testing. These robots are intended to study human-robotic interactions as well as test new autonomous capabilities in space. The Astrobee simulator and flight software were used in this study. We modified the flight software to follow a moving fiducial marker simulating a target placed on an astronaut's sleeve, a capability not currently available on the real Astrobees. The motivating scenario is that of an astronaut undergoing a work task and having the robot "hover" about a meter away to be ready to assist, e.g. by holding tools, recording the procedures, or cataloging parts. In addition to modifying the flight software, we model the dynamics of Astrobee using the Kane's method, and obtain a linear state-space realization of the system. Using this realization as our reference plant model, we develop novel adaptive robust MIMO control systems for the stowed and deployed robotic arm configurations of Astrobee, using the Youla Parameterization and H_∞-Optimization techniques. By performing uncertainty and robustness analysis on the system and running closed-loop simulations, we validate our control system design and demonstrate the merit in adopting a deterministic approach to control system design for safety-critical free-flying space robots.

1 Introduction

Missions on the International Space Station require great attention to detail to ensure no small procedure is missed. Keeping track of tools and supplies, for example, have led to some creative ways to manage everything. Some astronauts may attach tools to their uniforms or to the walls with velcro. Some parts or tools may be left suspended in zero-g while the astronaut tends to a sub-task. Throughout a task, an astronaut may need verbal, visual, or physical assistance while cognitively fully-loaded. In this instance, in order to keep proper situational awareness, a robot assistant could provide an offload of their routine work [1]. Astrobee, the most recent robot to be added to the station, autonomously navigates the modules of the ISS and performs tasks with astronauts or alone. We propose the following scenario for astronauts where Astrobee can help.

> An astronaut is performing repairs in node 2 when they realize that they are required to use a wrench to resize a portion of the EVA mobility unit (EMU) that is in the Japanese Experiment Module (JEM). Astrobee is tasked to undock, locate and grab the tool and assist the astronaut at their work-site. Astrobee retrieves the tool then station-keeps a short distance from the astronaut using a fiducial marker attached to the astronaut's sleeve, where it waits with the tool until the astronaut needs it. [1]

With this use-case, we will demonstrate an improved adaptive, robust controller with path and trajectory planning for following a fiducial marker. All simulation is done in ROS[2] using a public Astrobee repository[3]. Python is used to model the multibody dynamics of the Astrobee system with the robotic manipulator unfurled and grasping a tool, as depicted in the title page figure. MATLAB/Simulink is used to design the low-level control system.

[1] Scenario inspired by astronaut Steve Robinson's retelling of a common frustration aboard the ISS.
[2] https://www.ros.org/
[3] https://github.com/nasa/astrobee

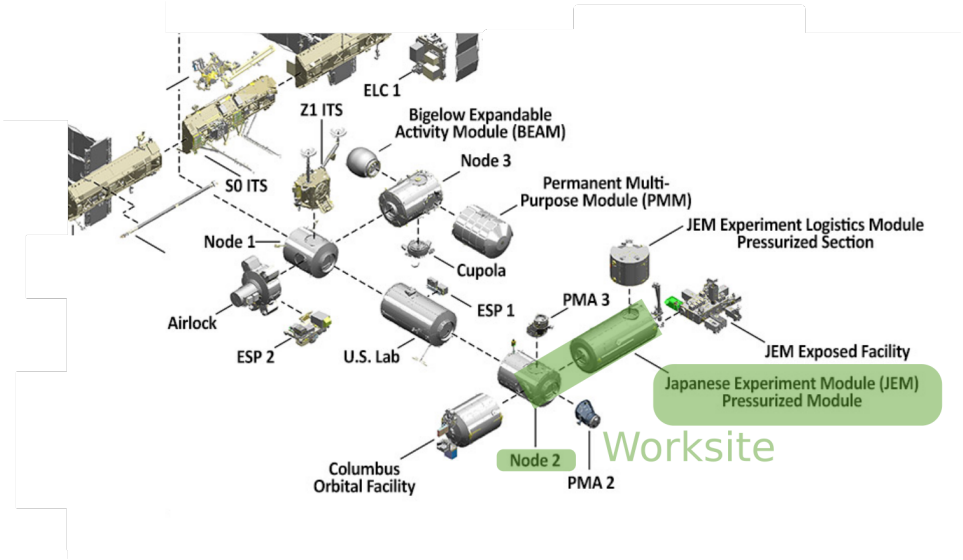


Figure 1: The proposed work-site for the Astrobee navigation experiment [2].

2 Literature Review

2.1 Astrobee Design

2.1.1 Design Overview

Astrobee are a new generation of free-flying robots on the International Space Station (ISS) that are built upon previous free-flyers such as the Synchronized Position Hold, Engage, Reorient, Experimental Satellite (SPHERES) and Personal Satellite Assistant (PSA). Bualat et al. [3] provide a history of previous ISS free-flyers, their uses and interactions with crew, and some limitations and considerations required for operation in the conditions of the ISS. One main distinguishing feature of Astrobee is that it is self-contained using vision-based navigation and able to operate in any segment of the ISS without the need for external wall-mounted cameras or sensors, unlike SPHERES. Astrobee has two propulsion modules mounted on the sides. Each module contains an impeller that draws air in from the side to charge a plenum that feeds six louvred vent nozzles per module. Figure 4 depicts the locations of the nozzles. As there is no expendable propellant, Astrobee is able to move as long as it has charged batteries. Charging and high-speed data transfer are done via a docking station mounted on the wall of the ISS that Astrobee docks with autonomously. Astrobee robots are equipped with the following hardware:

- High-, mid-, and low-level processors
- Twelve exhaust nozzles arranged two on each face, providing holonomic motion
- A manipulator arm used for perching on ISS handrails
- Forward and aft flashlights
- Six external sensors used for navigation, localization, and recording video
- Touchscreen
- Status LEDS
- WiFi connectivity

Smith et al. [4] provide design details and specifications of Astrobee including sensors, pose estimation, navigation, and control. The primary sensors are the inertial measurement units (IMU),

NavCam, DockCam, PerchCam (flash LIDAR depth sensor), and SpeedCam (sonar/optical flow sensor).

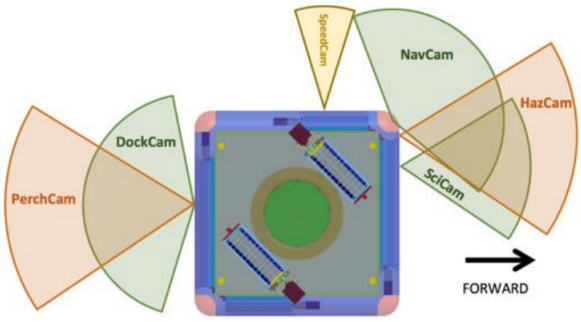


Figure 2: Location and FoV of Astrobee's six external sensors. [4]

Figure 2 depicts the locations and approximate fields-of-view (FoV) of Astrobee's six sensors. In the *general-purpose* mode, the NavCam and SpeedCam are used to match images with a prior map of features to provide absolute position. In the *fiducial-relative* mode, Astrobee matches AR targets detected by NavCam and DockCam with a map of the target locations to provide more accurate and robust position information during docking, as depicted in Figure 3. Additional details on the localization approach can be found in Coltin et al. [5]. For navigation and control, Astrobee either follows stored trajectories sent by ground controllers or is commanded by guest scientists. From the stored trajectory, repair trajectories are calculated, and a PID controller calculates the required forces and torques. From there, nozzle servo positions are calculated to provide the required thrust and are sent to a propulsion module controller (running in the *low-level processor*), which sends commands to the individual servos.

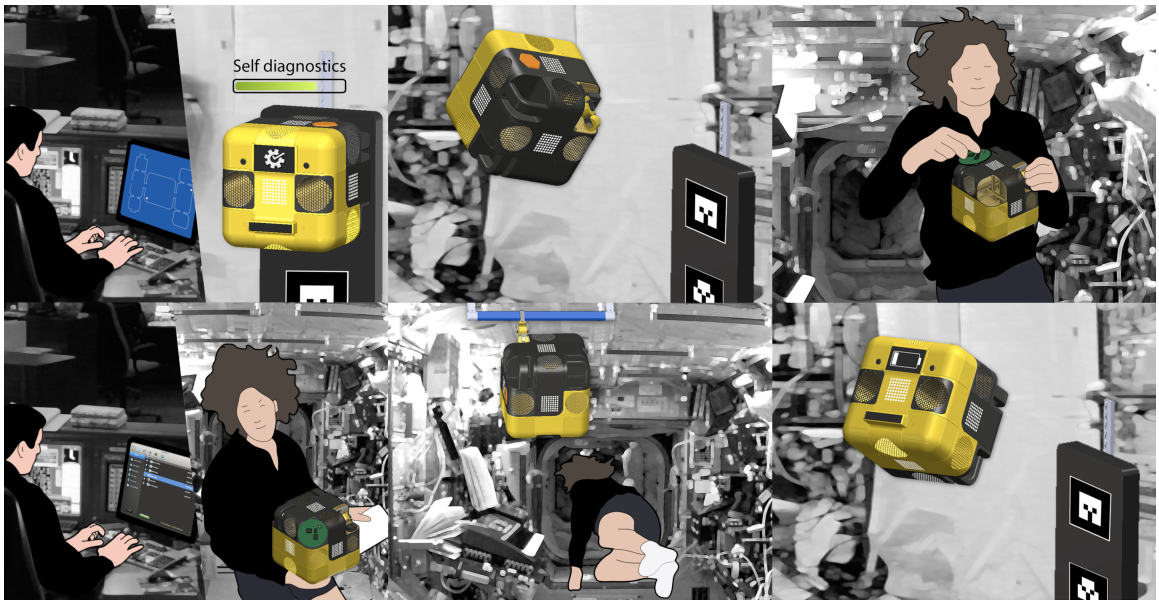


Figure 3: A scenario where Astrobee encounters fiducials while working with an astronaut [6].

2.1.2 Dynamics

Astrobee is a cubic robot with sides measuring about 30 cm across. Propulsion is provided by two fans located on the sides of the robot, and servo-actuated louvred vent nozzles, which allow for full six degree-of-freedom holonomic control. The twelve nozzles are arranged so that there are two on each face of the robot, and the air is directed to discharge perpendicular to the plane of the faces they are in, as depicted by Figure 4.

The dynamics of Astrobee can be determined by taking into account physical properties of the robot such as the location of the center of gravity, the impellers, vents, and the masses of any hosted payloads in the three payload bays of Astrobee. Watterson et al. [7] describe the system dynamics in detail, although the focus of their study was in developing a minimum-jerk smooth trajectory for obstacle avoidance while navigating through the ISS.

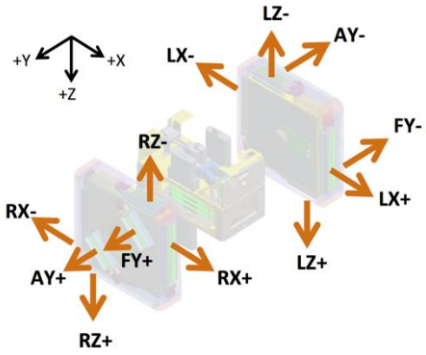


Figure 4: Exploded view of Astrobee showing names and locations of the nozzles. [4]

2.1.3 Control and Estimation

Astrobee uses a PID controller to compute the force and torque commands to correct the error between its current state (pose and velocity) and the output of the command shaper. The command shaper computes a short-term repair trajectory that smoothly transitions from the current state to the target state, while taking deviations from the target trajectory into account and satisfying operating constraints in the process.

Astrobee uses an augmented-state Extended Kalman Filter (EKF) for state estimation [4]. The filter fuses information from Astrobee's cameras depending on the navigation modes mentioned earlier. This includes image matching, point clouds, IMU, or optical flow measurements.

2.1.4 Robotic Manipulator: Perching Arm

Park et al. [8] describe the development of the 3-DoF perching arm for Astrobee. The arm includes a 1-DoF gripper, and a proximal and distal joint that allows Astrobee to pan and tilt while the gripper "perches" Astrobee to a handrail. While the use of the perching arm has been documented as possibly supporting manipulator research on the ISS [3, 4], the authors of this proposal could find no literature on the use of the Astrobee perching arm for carrying equipment.

2.2 Multibody Dynamics

The modeling of dynamical systems and robotic manipulators has been studied extensively in the literature. Classical methods such as the Newton-Euler method, Lagrangian formulations and Hamiltonian mechanics are popular, but can be cumbersome and computationally intensive for analytical dynamics modeling. Kane's method [10], on the other hand, is a highly systematic approach that becomes especially desirable in the multibody dynamics modeling of complex systems, such as the one depicted in Figure 5.

We will model Astrobee as a combination of cuboids for the main rigid body and simple linkages for the arm. The unknown tool will be modeled as a cylinder with dimensions and mass within reasonable bounds.

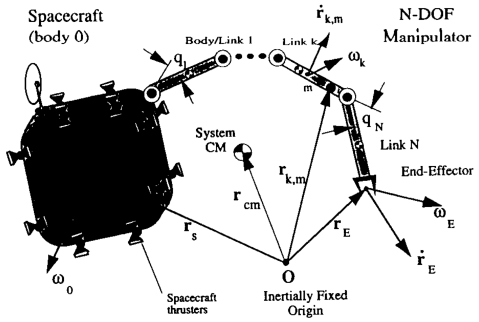


Figure 5: The generalized coordinates of a multibody free-flying space manipulator system [9]

2.3 Multivariable Adaptive Robust Control

PID controllers have been widely studied and are very popular, given the ease of design and implementation. However, these controllers usually suffer from the lack of robustness to system parameter uncertainty and external disturbances. This can pose a significant danger in safety-critical environments such as the ISS (especially in the Cupola and Airlock) [6]. Astrobee's control must come with guaranteed performance in the name of safety and reliability, and hence, must rely on more robust solutions for tasks involving grasping drifting objects with its perching arm, given the uncertainty in not only the properties of the object to be carried (mass, etc), but also the *way* in which the object is grasped. For example, a partial grasp of a tool could lead to uncertainty in the inertial properties of the combined system (moment of inertia, etc).

Many optimal control techniques such as the Linear Quadratic Regulator (LQR) and the Linear Quadratic Gaussian (LQG) have also been widely investigated in the literature for both single-input-single-output (SISO) and multiple-input-multiple-output (MIMO) systems. Although these techniques can be used for the control of multivariable processes, methods such as LQG have been proven to be non-optimal, while also lacking robustness to system parameter uncertainty [11]. This fact was exemplified by the failures of LQG controller implementations on two separate occasions in 1975: the controller on a Trident submarine caused it unexpectedly surface in a rough sea simulation. Although heuristic methods such as Loop Transfer Recovery (LTR) can increase the robustness of LQG controllers, they come at the price of severely degrading the original LQG cost-function, and thus lead to non-optimal solutions [12]. Methods such as nonlinear Model Predictive Control (MPC) are capable of handling complex nonlinear system dynamics, but they come at the cost of being computational intensive. This is certainly true for online implementations, given that the algorithms require a solution to an optimization problem at every sampling instant to obtain the necessary control commands [13].

Adaptive control is concerned with marked changes in a system plant. While knowledge about a system may not be exact, knowledge about how the system *changes* may be available. For example, an aircraft may perform well with a controller designed around its starting mass, but as it loses fuel, a new controller may be necessary to reach the required performance specifications. These controllers may be be fuzzy in that the controller changes depending on the system configuration [14]. In the UAV context, [15] has shown fuzzy logic for a path following controller. Alternatively, a range of linear controllers may be used to cover various operating points of a non-linear or time-varying plant. The linear controller is decided based on the system state and its closest operating point.

Given that there are strict safety-driven constraints on the achievable velocities and accelerations for free-flying robots inside the ISS, and that such robots usually travel at very low speeds without any sudden motion, linear control system design becomes increasingly desirable. Linear, constant-

gain, multivariable robust controllers can be designed offline and deployed for real-time control of systems, with high efficiency and low computational intensity. These designs can be extended to adapt to changing system parameters, thus making them adaptive and robust in nature. Methods such as the well-known H_∞-Optimization and the less-explored Youla Parameterization techniques provide a procedural framework for the design of efficient and reliable multivariable control systems, with guaranteed performance and stability robustness characteristics. Youla Parameterization-based control design has been investigated and successfully implemented in the automotive industry, especially for robust observer and estimation design [16, 17]. H_∞-Optimization-based control has been successfully deployed on the Ariane 5 Evolution launch vehicle for the atmospheric flight phase, replacing the previously used LQG controller. In telecommunication satellites, H_∞-Optimization-based control has been shown to reduce the propellant mass consumption by 10% during station-keeping maneuvers [18]. Multivariable adaptive robust control design techniques can ensure safe and robust robotic operations on the ISS, by adapting to different system configurations (eg. perching arm folded vs. deployed), while remaining robust to system parameter uncertainty and external disturbances in every possible configuration.

2.4 Planning Algorithms

An autonomous vehicle requires a set of instructions on where to go and when. The areas of path and trajectory planning have gained considerable traction upon the proliferation of UAVs and commercial robotics in general. Path planning considers the line or space along which a robot is required to travel while a trajectory includes information about the time in which it does so [19]. A higher-level controller can place constraints on the nature of how a robot moves along a path or trajectory. These position or timing constraints can be dependent on environmental geometry, such as obstacles or danger zones. In order to generate a given motion, the controller must provide the current state and the target state. Advanced trajectory-tracking controllers may also require knowledge about the vehicle dynamics to generate feasible plans.

2.4.1 Path Planning

In general, path planning is an easier task because there is no time requirement. Path-only convergence is accomplished with smaller control effort and less transient error [20]. With these advantages, obstacle avoidance may be more robust and effective, leading to a safer robot. Two simpler algorithms for path planning that are used on open-source flight controllers are the carrot-chasing algorithm [15] and the navigation vector field method [21]. These algorithms have the advantages of fast computation and broad applicability.

The carrot chasing algorithm uses a simple virtual path, sometimes constructed with waypoints, which is followed by a UAV by adjusting its heading toward a virtual point on the virtual path. The virtual point is a constant length from the vehicle to the path in the forward direction. By commanding this heading, the UAV will approach a tangent velocity to the virtual path. This method works well for straight line paths but shows decreasing performance for more complex shapes.

Navigational vector fields are constructed by developing vector valued functions that depend on the current position of the robot and return the direction in which it should travel. This kind of navigation can be termed as an attractive field method. This uses the analogy of a charged particle near an attractive or repulsive field. The motion of the particle can be predicted with knowledge of the vector field generated by the environment. Extending this analogy to a UAV, obstacles or walls would generate repulsive fields, while a desirable path would generate an attractive field [22]. Inside the ISS, Astrobee would assign walls and important equipment repulsive fields and with superposition, the sum of these fields would generate the entire vector field. When Astrobee is placed in the environment, the vector valued function returns the commanded direction vector which is passed to the lower level controller. Difficulties with this method include generating these

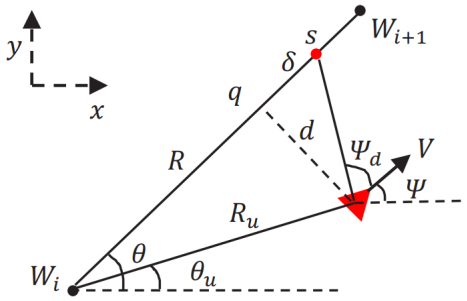


Figure 6: The vehicle is represented by the red triangle traveling with velocity V. The virtual path is piece-wise linear through the way-points, W_i. The algorithm generates a constant-length line path to the virtual point s and the heading commands a heading change proportional to Ψ_d [15]

vector fields on the fly and local minima.

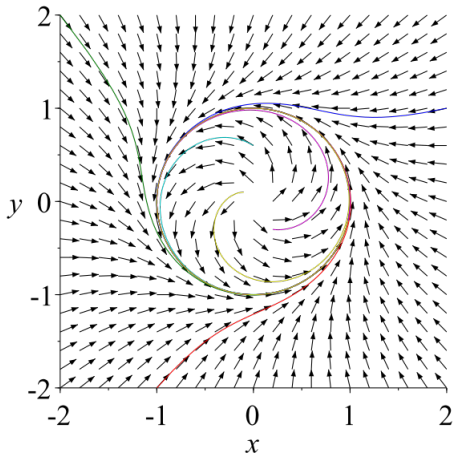


Figure 7: An example vector field for an orbital path around a central point. In this case, a circular vector field pushes a vehicle parallel to the vector at its current position [23]

3 ROS Implementation

The public repository of NASA's Astrobee is a comprehensive simulation of the robot's guidance, navigation and control set on the International Space Station [4]. Before deployment, NASA engineers tested in simulation, then on a granite slab, then on station. The granite slab and ISS were modeled with high precision in CAD to support these tests. This prior work enabled us to create a high fidelity look at what is going on in each subsystem and how these subsystems interact with the environment to aid scientific research on board. Following our scenario proposal, we began by commanding a fiducial marker to move about the Japanese Experiment Module (JEM), aka Kibo. To simulate a real astronaut doing work, we streamed a shoulder point trajectory from an HRVIP Space Ambulance experiment. This data was collected on a subject test in the interior layout of a Space Ambulance mock-up, so the trajectory is appears random, but holds important information

[4] https://github.com/nasa/astrobee

about the frequency of human movement. Thus, our marker following behavior must perform well with this type of noise. However, we made this point trajectory more interesting by moving the marker with this type of noise throughout the JEM and into the adjacent node.

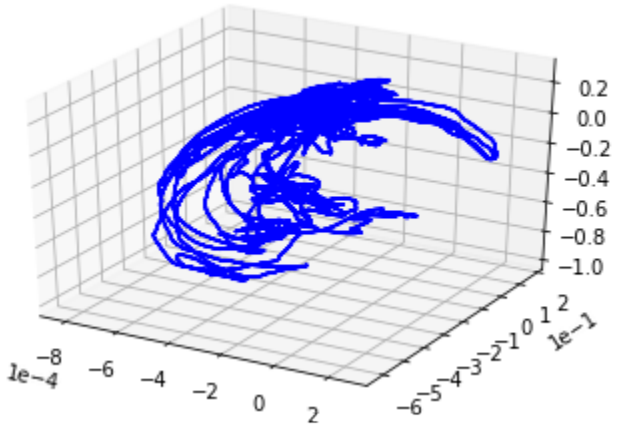
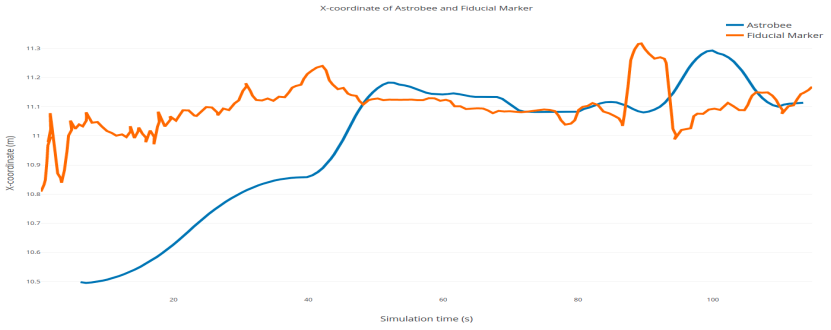


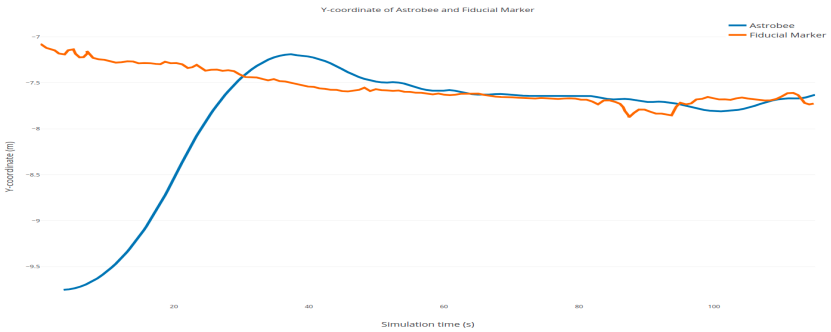
Figure 8: 3D Body motion data of the right shoulder

To assist the astronaut with work tasks, Astrobee must follow them to: inventory parts, film the astronaut's actions, and retrieve and hand off tools. For increased performance estimating position, i.e. during docking, Astrobee's visual localization system tracks the location of the marker with the robust Aruco fiducial tracking rather than feature detection as is normally done. Due to the complexity of this simulation system, we interfaced with the high-level position controller to command ideal positions near the marker for Astrobee to follow. We added an offset to the marker that keeps Astrobee away from where the astronaut would be positioned. This was accomplished using a virtual plane; the height of Astrobee's tracking position was held constant above the astronaut while the the horizontal axes (x and y) tracked the marker. The fact that Astrobee is always in the same plane makes it easier for the astronaut to anticipate what the robot is doing.

The results of this planner are shown in Figure 9. These show how Astrobee begins at its docking station (a large delta between Astrobee and marker, especially in the y direction) then approaches the marker and eventually settles to a position near the marker and station-keeps from above. Astrobee maintains its distance from the marker and actuates less frequently than a constantly updating follower. This is modeled by commanding the Astrobee to move to a location, then letting it arrive to this point before accepting a new move command. This was a restriction of the flight software (cannot interrupt a move command with a new location without first coming to a full stop) and was not able to be circumvented. That said, this behavior saves energy and is easy to anticipate. Future work would include more sophisticate path planning to interrupt the current path plan with a new one in response to new observation data, and to avoid some low-frequency movement. We would try to average these out and save power by only moving when the astronaut makes a longer-term adjustment in position. Such approaches could include an alpha beta filter. The need to accommodate changes in the astronaut's position as well as carry an unknown and unpredictable load of tools motivated the exploration of utilizing adaptive and robust controllers in the next sections. While they were not integrated into the full Astrobee flight software, these preliminary investigations show promise for integration as a replacement for the PID controller currently in Astrobee.



(a) x coordinates



(b) y coordinates

Figure 9: Comparison of x and y coordinates vs Time for Astrobee and the Fiducial Marker. Astrobee begins about 2.5 meters away from the marker.

4 Dynamics Modeling

We model the dynamics of Astrobee from scratch as described in the appendix, and obtain a linear state-space realization of the system. Based on the holonomic translation and attitude decoupling of Astrobee, the resulting combined transfer function matrix is split into individual translation and attitude plants in order to facilitate MIMO control system design.

5 Control System Design

Astrobee must possess guaranteed stability in the face of varying and unknown parameters to ensure the safety of the ISS and the astronauts aboard it. Astronauts typically need different kinds of tools with different properties for various aspects of their missions. Here, we have designed robust MIMO controllers to handle uncertainty in tool properties (mass, inertia tensor), which manifest as uncertain constants in the plant model. We have ensured that changing parameters that arise from these varying properties will not adversely affect performance or stability of the closed-loop system. As Astrobee is designed to possess full 6-DoF holnomic control, the system is decoupled in translation and attitude. Hence, two distinct controllers can be designed for these decoupled attitude and translation plant models.

We adopt three different control system design approaches for Astrobee with its perching arm in the stowed configuration. The first approach involves coordinate feedback only (position and attitude), and the holonomicity of the system allows for full decoupling of attitude and translation. We then design independent SISO controllers for each of the decoupled channels for the entire plant

using Youla Parameterization.

For the next two approaches, we consider full-state feedback (position, attitude, velocities and angular velocities). In these approaches, the controller for each input-output channel in the MIMO system is responsible for both position (or attitude) and velocity (or angular velocity) responses. As a result, although the translation and attitude components of the system are decoupled—translation and attitude themselves are not. Therefore, we model Astrobee as a double-plant system, with distinct plant models for translation and attitude.

In the second control system design approach, for each of the two plants, we employ the Youla Parameterization technique to design robust controllers, using a novel full-MIMO algebraic control design algorithm, as described in the appendix. In the third approach, we design robust controllers using H_∞-Optimization, by designing first-order weighting filters to shape the closed-loop frequency responses (the complementary sensitivity transfer function, T, and the sensitivity transfer function, S) and a constant-gain weighting filter to shape the actuator-effort frequency response (the Youla transfer function, Y).

We ran closed-loop simulations of the plant model(s) in conjunction with the designed internally-stable controllers, and in all the cases, the closed-loop system performed nominally, with acceptable overshoots and realistic settling times. Further, the closed-loop system displayed stability and performance robustness characteristics as well as robustness to uncertainties.

The characteristics of each response are evaluated by the shape of the associated singular value plots. Each controller we have designed satisfies these properties. The ideal shapes of these plots are listed below:

- $\sigma_{max}(S_y) \ll 0$ at low frequencies $\rightarrow$ Disturbance rejection
- $\sigma_{all}(T_y) = 1$ at low frequencies $\rightarrow$ Reference tracking
- $\sigma_{max}(T_y) \ll 0$ at high frequencies $\rightarrow$ Sensor noise, robust stability
- $\sigma_{max}(Y) \ll 0$ at high frequencies $\rightarrow$ Small control effort
- $\sigma_{min}(L) \gg 0$ at low frequencies $\rightarrow$ Disturbance rejection, reference tracking
- $\sigma_{max}(L) \ll 0$ at high frequencies $\rightarrow$ Sensor noise, robust stability
- $\sigma_{max}(G_c) \ll 0$ at high frequencies $\rightarrow$ Small control effort

5.1 Robust Control Design via Youla Parameterization

As the system (plant) is not BIBO stable with poles at the origin of the s-plane, we choose a Youla parameter, $Y(s)$, to cancel out the unstable poles in the plant. We make sure that the closed-loop step-responses are similar to that of a second-order Butterworth filter (equation 1). This is our design for a single channel of the decoupled system. For this design strategy, we choose a Youla parameter, Y, as follows:

$$Y(s) = \frac{Ks^2(\tau_z s + 1)}{(s^2 + 2\zeta\omega_n s + \omega_n^2)(\tau_p s + 1)(\tau_{px} s + 1)^2} \tag{1}$$

where,

$K = 1 \rightarrow$ gain, which will be chosen so as to satisfy the interpolation conditions (equation 2)
$\tau_p = 0.1 \rightarrow$ time-constant of the chosen pole
$\tau_{px} = 0.5 \rightarrow$ time-constant of the chosen pole to bring down the magnitude of the Youla frequency response at higher frequencies
$\zeta = \frac{1}{\sqrt{2}} \rightarrow$ damping ratio of the second order pole

$\omega_n = 1 \rightarrow$ natural frequency of the second order pole
τ_z (to be solved for) $\rightarrow$ time-constant of the chosen zero

The interpolation conditions to be satisfied in order to ensure internal stability of the closed-loop system in this case (unstable double-pole at s = 0) are as follows:

$$T(0) = 1 \quad \text{and} \quad \frac{\partial}{\partial s} T(0) = 0 \tag{2}$$

To achieve this, $\tau_z = 2.5142$ was chosen. Since the interpolation conditions are satisfied with this choice, we achieve good reference tracking. The other parameters were chosen by identifying safe response times in the context of the ISS. Since we do not require a very fast response, we can capitalize on other benefits of the control system, like noise and disturbance rejection. Hence, we chose the time constants as seen above.

The sensitivity transfer-function $S(s)$ is given by:

$$S(s) = 1 - T(s) \tag{3}$$

The controller (autopilot) for the translation plant model of Astrobee in the stowed configuration is computed by:

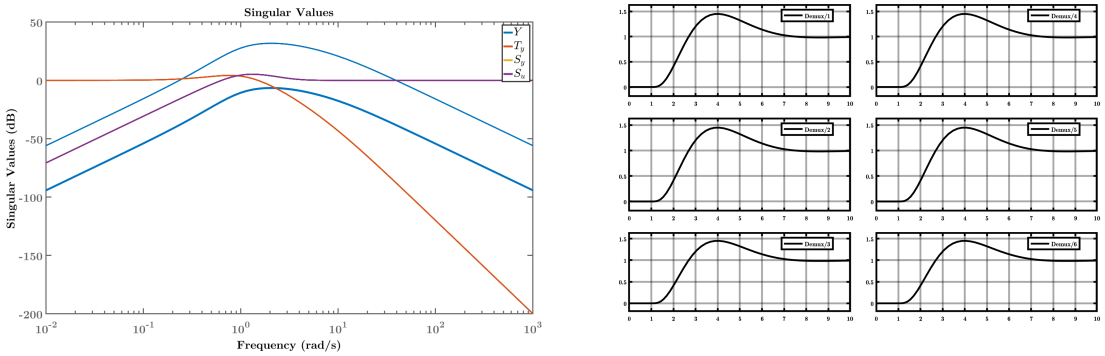
$$G_c(s) = \frac{Y(s)}{S(s)}$$

$$G_c(s) = \frac{635.1(2.5s + 1)}{(s^3 + 15.4s^2 + 64.8s + 116.2)} \tag{4}$$

The Youla parameter, $Y(s)$, the sensitivity transfer function, $S(s)$, the complementary-sensitivity transfer function, $T(s)$ and the product $G_p(s)S(s)$ are all BIBO stable. Thus, the conditions for internal stability are satisfied.

5.2 MIMO Control Design (Stowed Robotic Arm Configuration)

5.2.1 Youla Parameterization with Coordinate Feedback

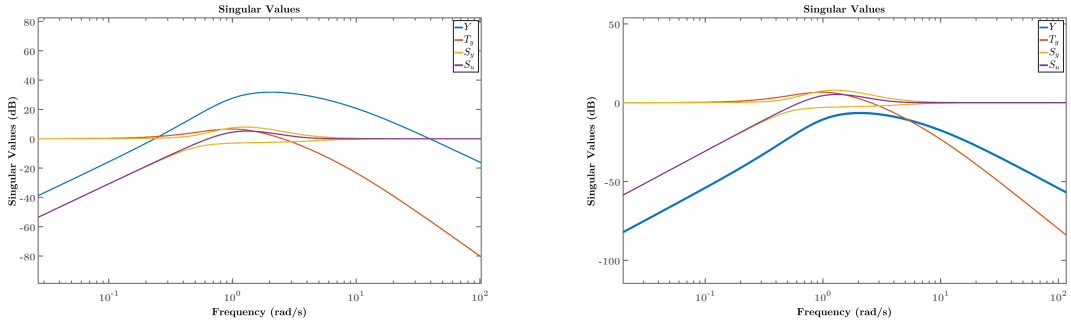


(a) Closed-loop frequency responses

(b) Closed-loop step responses

Figure 10: The closed-loop responses of the fully decoupled (diagonal) plant model in conjuction with the designed controller using Youla Parameterization

5.2.2 Youla Parameterization with Full-State Feedback



(a) Closed-loop frequency responses of the translation plant with the designed MIMO Youla translation controller

(b) Closed-loop frequency responses of the attitude plant with the designed MIMO Youla attitude controller

Figure 11: The closed-loop responses of the translation and attitude plant models in conjuction with the corresponding MIMO controllers designed using Youla Parameterization

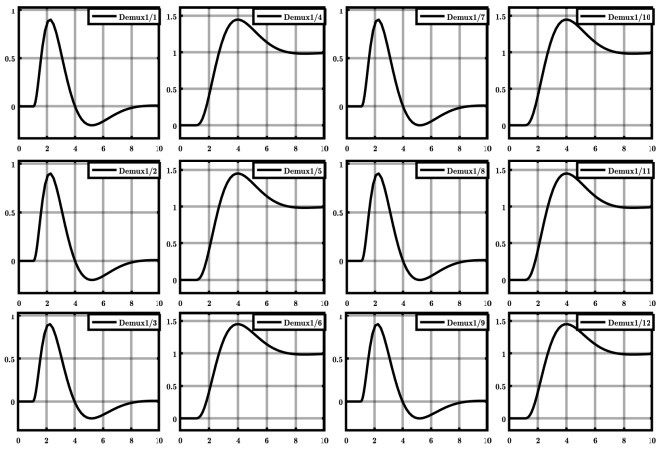
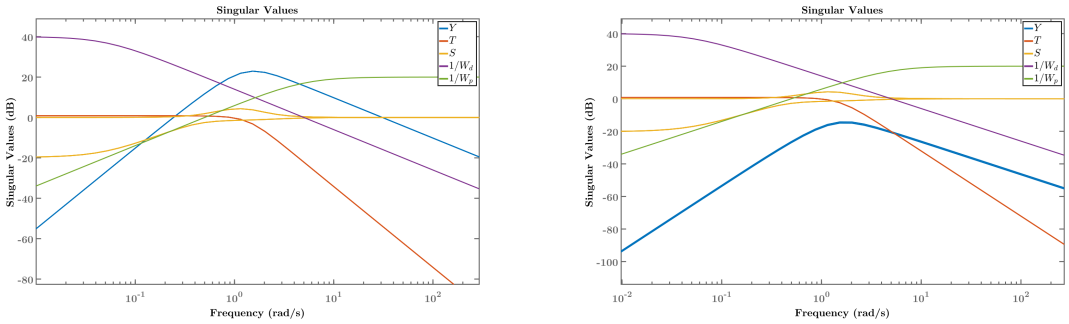


Figure 12: Closed-loop step responses of the combined translation and attitude plant model with the MIMO Youla controller—the four columns represent the velocity, position, angular velocity, and attitude step responses, respectively

5.2.3 H_∞-Optimization with Full-State Feedback



(a) Closed-loop frequency responses of the translation plant with the designed H_∞ translation controller

(b) Closed-loop frequency responses of the attitude plant with the designed H_∞ attitude controller

Figure 13: The closed-loop responses of the translation and attitude plant models in conjuction with the corresponding MIMO controllers designed using H_∞-Optimization

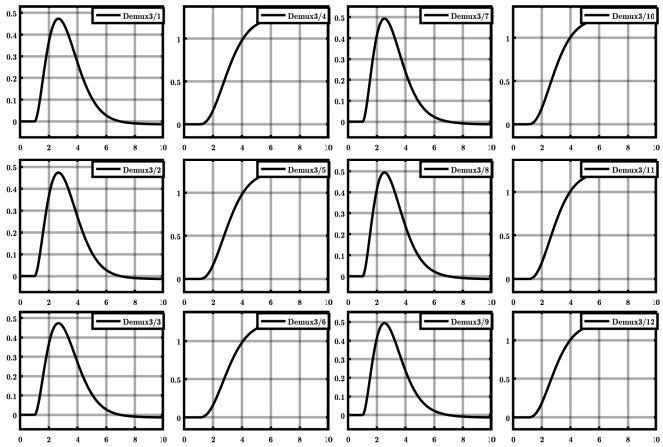


Figure 14: Closed-loop step responses of the the combined translation and attitude plant model with the MIMO H_∞ controller—the four columns represent the velocity, position, angular velocity, and attitude step responses, respectively

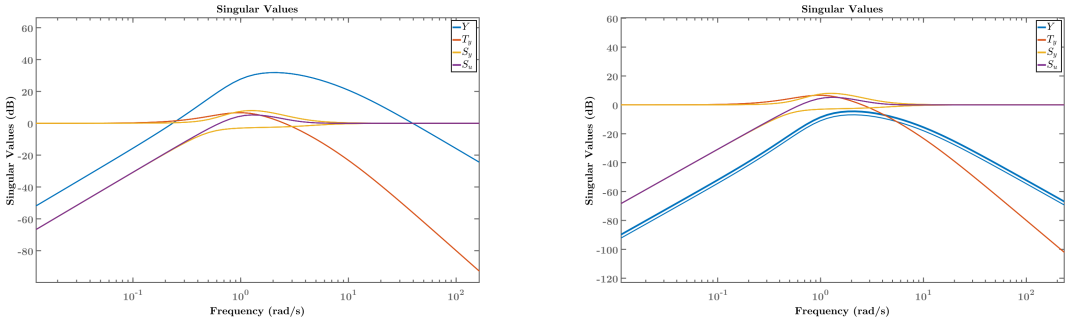
5.3 Adaptive Robust Control (Deployed Robotic Arm Configuration)

Although uncertainty may arise from the way Astrobee carries a tool or the uncertain properties of the tools being grasped themselves, the operation of grasping a tool itself would be predetermined. By adapting to the new configuration (deployed robotic arm grasping a tool), we can achieve better performance than by merely implementing an all-configuration constant-gain controller. This adaptation would involve switching between controllers based on the configuration.

Here, we design a MIMO robust controller using Youla Parameterization for the deployed robotic arm configuration of Astrobee, and analyze the robustness of the closed-loop system to parameter

uncertainty (uncertain tool properties → uncertain mass and moments of inertia → uncertain constants/gains in the plant model).

By performing an uncertainty analysis on the gains of the plant model and testing the stability and performance of the controller and the closed-loop system to step inputs, we validate our MIMO robust controller design.



(a) Closed-loop frequency responses of the translation plant (deployed) with the designed MIMO Youla translation controller

(b) Closed-loop frequency responses of the attitude plant (deployed) with the designed MIMO Youla attitude controller

Figure 15: The closed-loop responses of the translation and attitude plant models (deployed) in conjuction with the corresponding MIMO controllers designed using Youla Parameterization

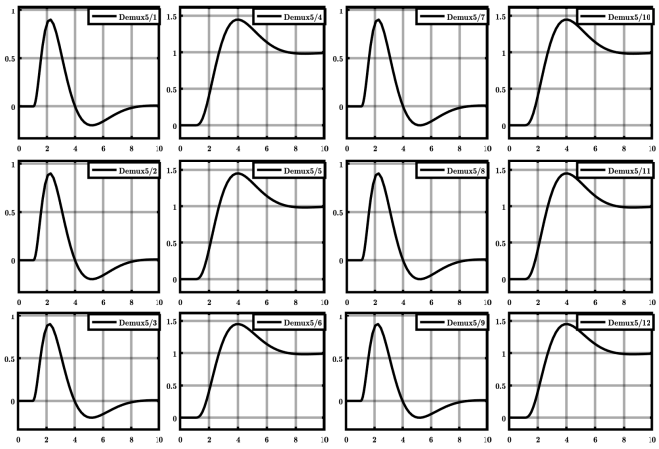
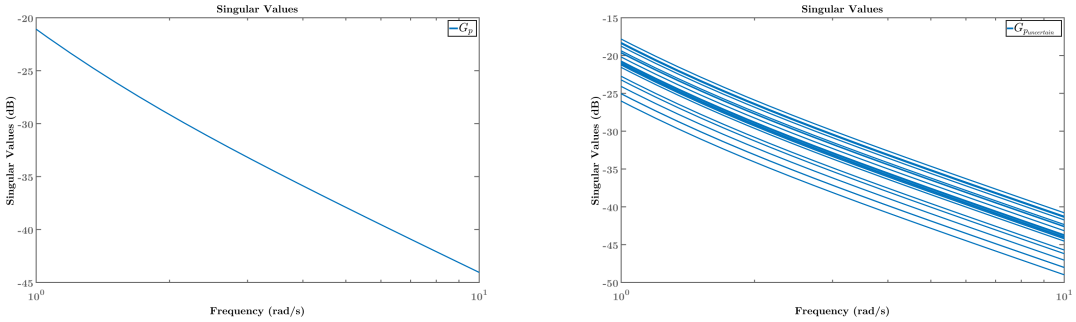


Figure 16: Closed-loop step responses of the the combined translation and attitude plant model (deployed) with the MIMO Youla controller—the four columns represent the velocity, position, angular velocity, and attitude step responses, respectively

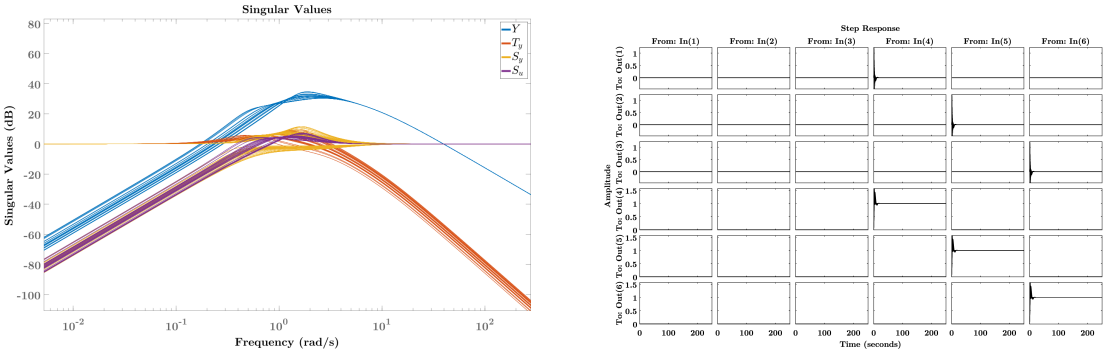
5.4 Robustness Analysis



(a) Frequency response of the deployed-configuration translation plant transfer function matrix

(b) Frequency response of the deployed-configuration translation plant transfer function matrix with uncertain gains

Figure 17: The frequency responses of the deployed-configuration translation plant transfer function matrix with and without gain uncertainty (50%), respectively



(a) Closed-loop frequency responses of the deployed-configuration translation plant model with 50% gain uncertainty

(b) Closed-loop step responses of the deployed-configuration translation plant model with 50% gain uncertainty

Figure 18: The closed-loop responses of the uncertain deployed-configuration translation plant model in conjunction with the robust MIMO Youla controller designed for the nominal translation plant—the closed-loop system is stable and nominally performant

6 Conclusion

We showed that Astrobee can follow an astronaut that is wearing a marker in order to provide support. We also showed that a new controller on Astrobee is robust to model variation from grasping different tools. Astrobee is a remarkable advance in collaborative robotics. With the ability to autonomously move about the International Space Station, the robot can assist astronauts to improve productivity and safety. As space stations evolve to more autonomous systems, without human intervention, we must assure that the autonomous agents can operate within safe bounds. By using robust controllers, we can rest easy knowing that model variation will have little effect on Astrobee's control system.

7 References

[1] J. L. Broyan, M. K. Ewert, and P. W. Fink, "Logistics reduction technologies for exploration missions," in *AIAA SPACE 2014 Conference and Exposition*, 2014, p. 4334.

[2] NASA, "International space station facts and figures," 2019. [Online]. Available: https://www.nasa.gov/feature/facts-and-figures

[3] M. G. Bualat, T. Smith, E. E. Smith, T. Fong, and D. Wheeler, "Astrobee: A new tool for iss operations," in *2018 SpaceOps Conference*, 2018, p. 2517.

[4] T. Smith, J. Barlow, M. Bualat, T. Fong, C. Provencher, H. Sanchez, and E. Smith, "Astrobee: A new platform for free-flying robotics on the international space station," 2016.

[5] B. Coltin, J. Fusco, Z. Moratto, O. Alexandrov, and R. Nakamura, "Localization from visual landmarks on a free-flying robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4377–4382.

[6] M. Bualat, J. Barlow, T. Fong, C. Provencher, and T. Smith, "Astrobee: Developing a free-flying robot for the international space station," in *AIAA SPACE 2015 Conference and Exposition*, 2015, p. 4643.

[7] M. Watterson, T. Smith, and V. Kumar, "Smooth trajectory generation on se (3) for a free flying space robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 5459–5466.

[8] I.-W. Park, T. Smith, H. Sanchez, S. W. Wong, P. Piacenza, and M. Ciocarlie, "Developing a 3-dof compliant perching arm for a free-flying robot on the international space station," in *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2017, pp. 1135–1141.

[9] S. Dubowsky and E. Papadopoulos, "The kinematics, dynamics, and control of free-flying and free-floating space robotic systems," *IEEE Transactions on robotics and automation*, vol. 9, no. 5, pp. 531–543, 1993.

[10] T. R. Kane and D. A. Levinson, *Dynamics, theory and applications*. McGraw Hill, 1985.

[11] J. C. Doyle, "Guaranteed margins for lqg regulators," *IEEE Transactions on automatic Control*, vol. 23, no. 4, pp. 756–757, 1978.

[12] D. Noll, "A generalization of the linear quadratic gaussian loop transfer recovery procedure (lqg/ltr)."

[13] C. Liu, W.-H. Chen, and J. Andrews, "Tracking control of small-scale helicopters using explicit nonlinear mpc augmented with disturbance observers," *Control Engineering Practice*, vol. 20, no. 3, pp. 258–268, 2012.

[14] J. D. Boskovic and R. K. Mehra, "Stable multiple model adaptive flight control for accommodation of a large class of control effector failures," in *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*, vol. 3, 1999, pp. 1920–1924 vol.3.

[15] S. A. H. Tabatabaei, A. Yousefi-koma, M. Ayati, and S. S. Mohtasebi, "Three dimensional fuzzy carrot-chasing path following algorithm for fixed-wing vehicles," in *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*. IEEE, 2015, pp. 784–788.

[16] F. Assadian, A. K. Beckerman, and J. Velazquez Alcantar, "Estimation design using youla parametrization with automotive applications," *Journal of Dynamic Systems, Measurement, and Control*, vol. 140, no. 8, 2018.

[17] Z. Liu, F. Assadian, and K. Mallon, "Vehicle yaw rate and sideslip estimations: A comparative analysis of siso and mimo youla controller output observer, linear and nonlinear kalman filters, and kinematic computation," *Journal of Mechanical and Aerospace Engineering*, vol. 1, no. 1, 2019.

[18] C. Philippe, A. Annaswamy, G. Balas, J. Bals, S. Garg, A. Knoll, K. Krishnakumar, M. Maroni, R. Osterhuber, and Y. C. Yeh, "Apollo as the catalyst for control technology."

[19] B. Rubí, R. Pérez, and B. Morcego, "A survey of path following control strategies for uavs focused on quadrotors," *Journal of Intelligent & Robotic Systems*, pp. 1–25, 2019.

[20] D. Cabecinhas, R. Cunha, and C. Silvestre, "Rotorcraft path following control for extended flight envelope coverage," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE, 2009, pp. 3460–3465.

[21] H. G. De Marina, Y. A. Kapitanyuk, M. Bronz, G. Hattenberger, and M. Cao, "Guidance algorithm for smooth trajectory tracking of a fixed wing uav flying in wind flows," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 5740–5745.

[22] Y.-b. Chen, G.-c. Luo, Y.-s. Mei, J.-q. Yu, and X.-l. Su, "Uav path planning using artificial potential field method updated by optimal control theory," *International Journal of Systems Science*, vol. 47, no. 6, pp. 1407–1420, 2016.

[23] Yapparina, "File:vector field and trajectories of a simple limit cycle.svg," 2018. [Online]. Available: https://commons.wikimedia.org/wiki/File:Vector_field_and_trajectories_of_a_simple_limit_cycle.svg

8 Appendix

I. Tammer

a. Scenario development

b. Dynamics Modeling of Astrobee

c. Operating Astrobee simulator through various scenarios (baseline with arm stowed, arm unstowed, following Astronaut fiducial marker, etc)

II. Jason

a. Reactive path planning

b. Astrobee model description variation

c. Multivariable Adaptive Robust Control Design: Youla Parameterization, H_∞-Optimization

III. Abhi

a. Dynamics Modeling of Astrobee

b. Plant Linearization → State-Space Realization → MIMO Transfer-Function Modeling

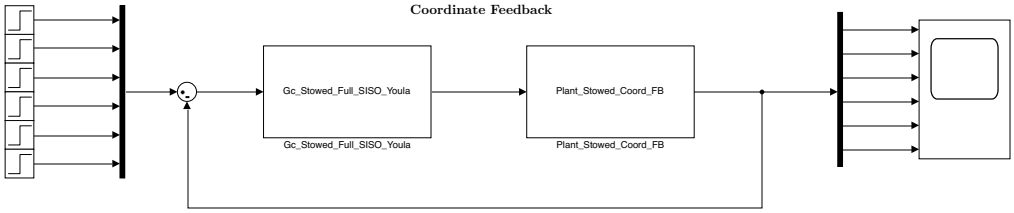
c. Multivariable Adaptive Robust Control Design: Youla Parameterization, H_∞-Optimization

d. Robustness and Performance Analysis of the Closed-Loop System; Comparison

Algorithm Multivariable Feedback Control Design via Youla Parameterization

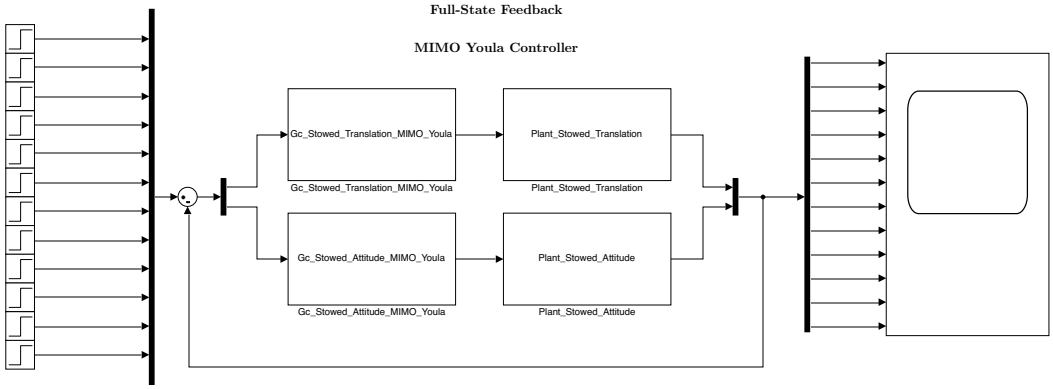
procedure MIMO(G_p) ▷ Plant TF
  compute the SM-form of G_p, M_p ▷ Smith-McMillan form
  compute an MFD of G_p, $N_R {D_R}^{-1}$ ▷ Matrix Fraction Description
  find unimodular U_L, $U_R \mid M_p = U_L G_p U_R$
  choose M_Y to shape $M_T = M_p M_Y \mid M_T \to$ diagonal ▷ M_T decouples T_y
  check: $Y = U_R M_Y U_L \to$ all entries of Y are proper TFs ▷ Youla TF
  for k in $M_T(k,k) \mid M_T(k,k) = G_k$ **do**
    SISO(G_k) ▷ Internal Stability
  end for
  compute $T_y = U_L^{-1} M_p M_Y U_L$ ▷ Complementary-Sensitivity TF
  compute $S_y = I - T_y = U_L^{-1}(I - M_p M_Y) U_L$ ▷ Sensitivity TF
  compute $G_c = U_R(I - M_Y M_p)^{-1} M_Y U_L$ ▷ Controller TF
  check: $S_y G_p = U_L^{-1}(I - M_p M_Y) M_p U_R^{-1} \to$ meets requirement ▷ Loop-Shaping Check
  check: κ for G_p, $G_c \to$ satisfactory for all nodes ▷ Condition Number Check
  return G_c
end procedure
function SISO(G) ▷ SISO Interpolation Conditions
  if G has a $j\omega$-axis or RHP pole p of multiplicity a_p **then** ▷ Unstable Poles
    $S(p) = 0; T(p) = 1$
    for $j = 1 : a_p - 1$ **do**
      $\frac{d^j S(p)}{ds^j} = 0;\ \frac{d^j T(p)}{ds^j} = 0$
    end for
  end if
  if G has an NMP zero z of multiplicity a_z **then** ▷ Non-Minimum Phase Zeros
    $S(z) = 1; T(z) = 0$
    for $j = 1 : a_z - 1$ **do**
      $\frac{d^j S(z)}{ds^j} = 0;\ \frac{d^j T(z)}{ds^j} = 0$
    end for
  end if
end function

Stowed Configuration:
Coordinate Feedback



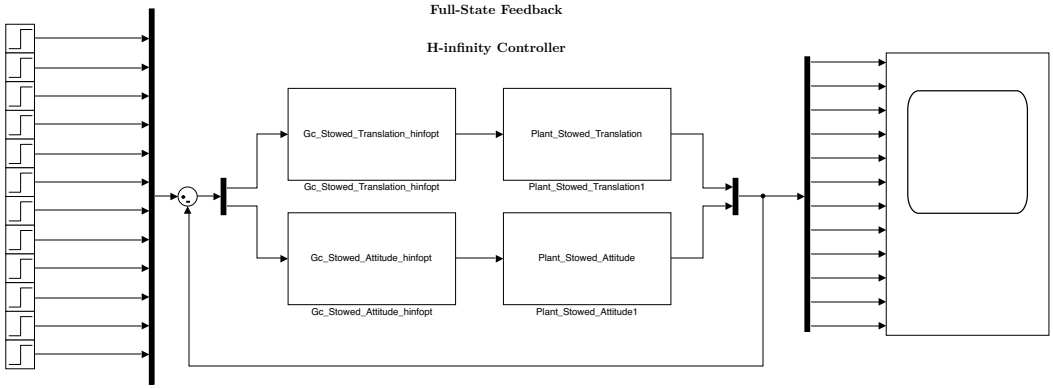
Stowed Configuration:
Full-State Feedback

MIMO Youla Controller



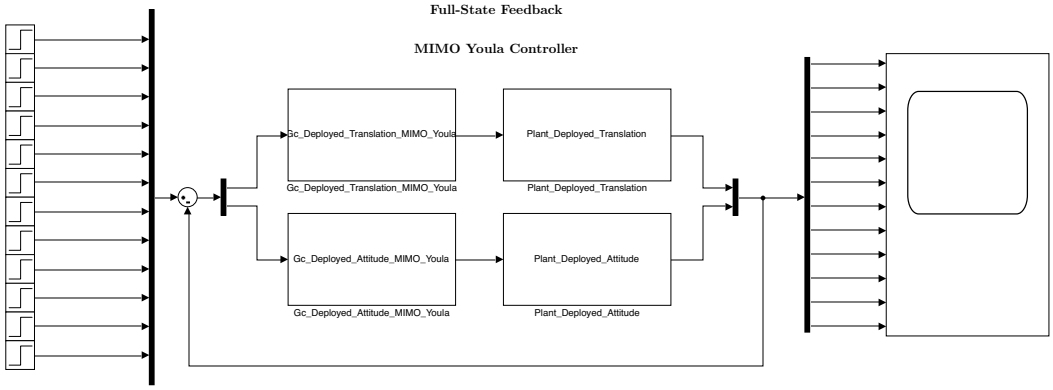
Stowed Configuration:
Full-State Feedback

H-infinity Controller



Deployed Configuration:
Full-State Feedback

MIMO Youla Controller



Astrobee_Euler

June 11, 2020

Modeling and Simulation of Astrobee

0.1 Configuration

Available configurations:

- "original" → models Astrobee in the stowed configuration with the physical parameters listed by NASA
- "stowed" → models Astrobee with its robotic arm stowed
- "deployed" → models Astrobee with its robotic arm deployed and holding a tool

[1]:
```
configuration = "deployed"
```

0.2 Dependencies

[2]:
```
import sympy as sm
import sympy.physics.mechanics as me
from pydy.system import System
import numpy as np
import matplotlib.pyplot as plt
from pydy.codegen.ode_function_generators import generate_ode_function
from scipy.integrate import odeint
import scipy.io as sio
me.init_vprinting()
```

0.3 Reference Frames

[3]:
```
ISS = me.ReferenceFrame('N') # ISS RF
B = me.ReferenceFrame('B') # body RF

q1, q2, q3 = me.dynamicsymbols('q1:4') # attitude coordinates (Euler angles)

B.orient(ISS, 'Body', (q1, q2, q3), 'xyz') # body RF
```

[4]:
```
t = me.dynamicsymbols._t
```

0.4 Significant Points

[5]:
```
O = me.Point('O') # fixed point in the ISS
O.set_vel(ISS, 0)
```

[6]:
```
x, y, z = me.dynamicsymbols('x, y, z') # translation coordinates (position of the␣
 ↪mass-center of Astrobee relative to 'O')
l = sm.symbols('l') # length of Astrobee (side of cube)
```

[7]:
```
C = O.locatenew('C', x * ISS.x + y * ISS.y + z * ISS.z) # Astrobee CM
```

0.5 Kinematical Differential Equations

[8]:
```
ux = me.dynamicsymbols('u_x')
uy = me.dynamicsymbols('u_y')
uz = me.dynamicsymbols('u_z')
u1 = me.dynamicsymbols('u_1')
u2 = me.dynamicsymbols('u_2')
u3 = me.dynamicsymbols('u_3')
```

[9]:
```
z1 = sm.Eq(ux, x.diff())
z2 = sm.Eq(uy, y.diff())
z3 = sm.Eq(uz, z.diff())
z4 = sm.Eq(u1, q1.diff())
z5 = sm.Eq(u2, q2.diff())
z6 = sm.Eq(u3, q3.diff())
u = sm.solve([z1, z2, z3, z4, z5, z6], x.diff(), y.diff(), z.diff(), q1.diff(), q2.
 ↪diff(), q3.diff())
u
```

[9]:
$$\left\{\dot{q}_1 : u_1,\ \dot{q}_2 : u_2,\ \dot{q}_3 : u_3,\ \dot{x} : u_x,\ \dot{y} : u_y,\ \dot{z} : u_z\right\}$$

[10]:
```
# ux_dot = me.dynamicsymbols('u_x_d')
# uy_dot = me.dynamicsymbols('u_y_d')
# uz_dot = me.dynamicsymbols('u_z_d')
# u1_dot = me.dynamicsymbols('u_1_d')
# u2_dot = me.dynamicsymbols('u_2_d')
# u3_dot = me.dynamicsymbols('u_3_d')
```

[11]:
```
# z1d = sm.Eq(ux_dot, ux.diff())
# z2d = sm.Eq(uy_dot, uy.diff())
# z3d = sm.Eq(uz_dot, uz.diff())
# z4d = sm.Eq(u1_dot, u1.diff())
# z5d = sm.Eq(u2_dot, u2.diff())
# z6d = sm.Eq(u3_dot, u3.diff())
# ud = sm.solve([z1d, z2d, z3d, z4d, z5d, z6d], ux.diff(), uy.diff(), uz.diff(), u1.
 ↪diff(), u2.diff(), u3.diff())
# ud
```

0.6 Translational Motion

0.6.1 Velocity

[12]:
```
C.set_vel(ISS, C.pos_from(O).dt(ISS).subs(u))
V_B_ISS_ISS = C.vel(ISS)
V_B_ISS_ISS # "velocity of Astrobee CM w.r.t ISS RF expressed in ISS RF"
```

[12]:
$$u_x\hat{\mathbf{n}}_\mathbf{x} + u_y\hat{\mathbf{n}}_\mathbf{y} + u_z\hat{\mathbf{n}}_\mathbf{z}$$

0.6.2 Acceleration

[13]:
```
A_B_ISS_ISS = C.acc(ISS).subs(u) #.subs(ud)
A_B_ISS_ISS # "acceleration of Astrobee CM w.r.t ISS RF expressed in ISS RF"
```

[13]:
$$\dot{u}_x\hat{\mathbf{n}}_\mathbf{x} + \dot{u}_y\hat{\mathbf{n}}_\mathbf{y} + \dot{u}_z\hat{\mathbf{n}}_\mathbf{z}$$

0.7 Angular Motion

0.7.1 Angular Velocity

[14]:
```
B.set_ang_vel(ISS, B.ang_vel_in(ISS).subs(u))
Omega_B_ISS_B = B.ang_vel_in(ISS)
Omega_B_ISS_B # "angular velocity of body RF w.r.t ISS RF expressed in body RF"
```

[14]:
$$(u_1\cos(q_2)\cos(q_3) + u_2\sin(q_3))\hat{\mathbf{b}}_\mathbf{x} + (-u_1\sin(q_3)\cos(q_2) + u_2\cos(q_3))\hat{\mathbf{b}}_\mathbf{y} + (u_1\sin(q_2) + u_3)\hat{\mathbf{b}}_\mathbf{z}$$

0.7.2 Angular Acceleration

[15]:
```
Alpha_B_ISS_B = B.ang_acc_in(ISS).subs(u) #.subs(ud)
Alpha_B_ISS_B # "angular acceleration of body RF w.r.t ISS RF expressed in body RF"
```

[15]:
$$(-u_1u_2\sin(q_2)\cos(q_3) - u_1u_3\sin(q_3)\cos(q_2) + u_2u_3\cos(q_3) + \sin(q_3)\dot{u}_2 + \cos(q_2)\cos(q_3)\dot{u}_1)\hat{\mathbf{b}}_\mathbf{x} + (u_1u_2\sin(q_2)\sin(q_3) - u_1u_3\cos(q_2)\cos(q_3) - u_2u_3\sin(q_3) - \sin(q_3)\cos(q_2)\dot{u}_1 + \cos(q_3)\dot{u}_2)\hat{\mathbf{b}}_\mathbf{y} + (u_1u_2\cos(q_2) + \sin(q_2)\dot{u}_1 + \dot{u}_3)\hat{\mathbf{b}}_\mathbf{z}$$

0.8 Mass and Inertia

[16]:
```
m = sm.symbols('m') # Astrobee mass

Ix, Iy, Iz = sm.symbols('I_x, I_y, I_z') # principal moments of inertia

I = me.inertia(B, Ix, Iy, Iz) # inertia dyadic
I
```

[16]:
$$I_x\hat{\mathbf{b}}_\mathbf{x}\otimes\hat{\mathbf{b}}_\mathbf{x} + I_y\hat{\mathbf{b}}_\mathbf{y}\otimes\hat{\mathbf{b}}_\mathbf{y} + I_z\hat{\mathbf{b}}_\mathbf{z}\otimes\hat{\mathbf{b}}_\mathbf{z}$$

0.9 Loads

0.9.1 Forces

[17]:
```
Fx_mag, Fy_mag, Fz_mag = me.dynamicsymbols('Fmag_x, Fmag_y, Fmag_z')

Fx = Fx_mag * ISS.x
Fy = Fy_mag * ISS.y
Fz = Fz_mag * ISS.z

Fx, Fy, Fz
```

[17]:
$$\left(|F|_x\,\hat{\mathbf{n}}_\mathbf{x},\ |F|_y\,\hat{\mathbf{n}}_\mathbf{y},\ |F|_z\,\hat{\mathbf{n}}_\mathbf{z}\right)$$

0.9.2

[18]:
```
Tx_mag, Ty_mag, Tz_mag = me.dynamicsymbols('Tmag_x, Tmag_y, Tmag_z')

Tx = Tx_mag * B.x
Ty = Ty_mag * B.y
Tz = Tz_mag * B.z

Tx, Ty, Tz
```

[18]:
$$\left(|T|_x \mathbf{\hat{b}_x},\ |T|_y \mathbf{\hat{b}_y},\ |T|_z \mathbf{\hat{b}_z}\right)$$

0.10 Kane's Method

[19]:
```
kdes = [z1.rhs - z1.lhs, z2.rhs - z2.lhs, z3.rhs - z3.lhs, z4.rhs - z4.lhs, z5.rhs -␣
↪z5.lhs, z6.rhs - z6.lhs]
```

[20]:
```
body = me.RigidBody('body', C, B, m, (I, C))
bodies = [body]
```

[21]:
```
loads = [(C, Fx),
         (C, Fy),
         (C, Fz),
         (B, Tx),
         (B, Ty),
         (B, Tz)
        ]
```

[22]:
```
kane = me.KanesMethod(ISS, (x, y, z, q1, q2, q3), (ux, uy, uz, u1, u2, u3),␣
↪kd_eqs=kdes)
```

[23]:
```
fr, frstar = kane.kanes_equations(bodies, loads=loads)
```

[24]:
```
# fr
```

[25]:
```
# frstar
```

0.11 Simulation

[26]:
```
sys = System(kane)
```

[27]:
```
sys.constants_symbols
```

[27]:
$$\{I_x, I_y, I_z, m\}$$

[28]:
```
if configuration == "original":
    sys.constants = {Ix: 0.1083,
                     Iy: 0.1083,
                     Iz: 0.1083,
                     m: 7
                     }
```

```
elif configuration == "stowed":
    sys.constants = {Ix: 0.185,
                     Iy: 0.202,
                     Iz: 0.188,
                     m: 15.878
                     }

elif configuration == "deployed":
    sys.constants = {Ix: 0.186,
                     Iy: 0.253,
                     Iz: 0.237,
                     m: 16.029
                     }
```

[29]:
```
sys.constants
```

[29]: $\{I_x : 0.186,\ I_y : 0.253,\ I_z : 0.237,\ m : 16.029\}$

[30]:
```
sys.times = np.linspace(0.0, 50.0, num=1000)
```

[31]:
```
sys.coordinates
```

[31]: $[x,\ y,\ z,\ q_1,\ q_2,\ q_3]$

[32]:
```
sys.speeds
```

[32]: $[u_x,\ u_y,\ u_z,\ u_1,\ u_2,\ u_3]$

[33]:
```
sys.states
```

[33]: $[x,\ y,\ z,\ q_1,\ q_2,\ q_3,\ u_x,\ u_y,\ u_z,\ u_1,\ u_2,\ u_3]$

[34]:
```
sys.initial_conditions = {x: 0.0,
                          y: 0.0,
                          z: 0.0,
                          q1: 0.0,
                          q2: 0.0,
                          q3: 0.0,
                          ux: 10.0,
                          uy: 0.0,
                          uz: 0.0,
                          u1: 0.0,
                          u2: 0.0,
                          u3: 0.0
                         }
```

[35]:
```
sys.specifieds_symbols
```

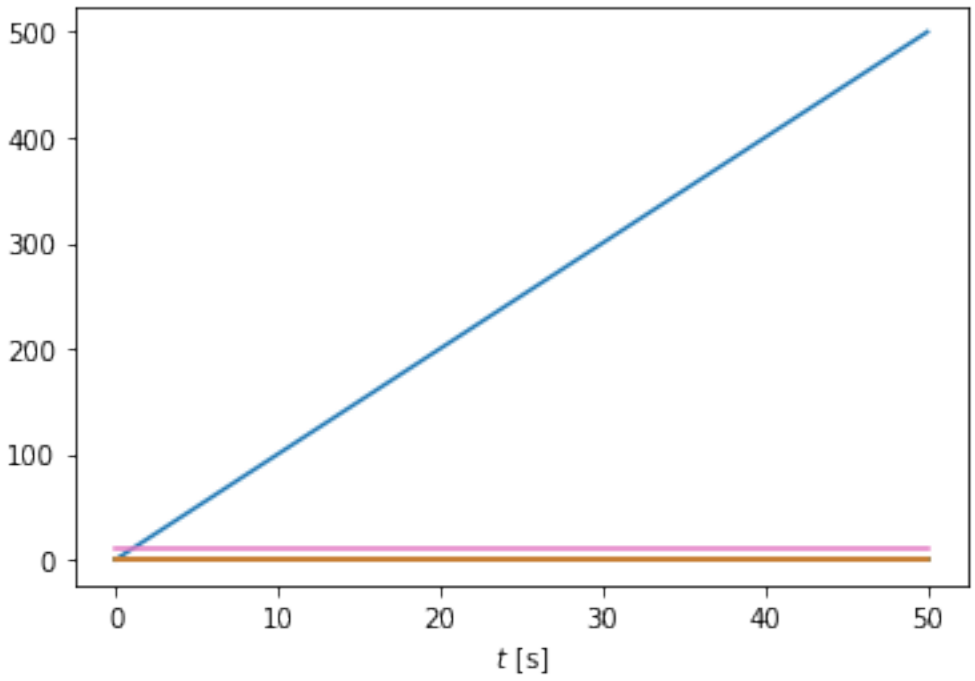
[35]: $\left\{|F|_x, |F|_y, |F|_z, |T|_x, |T|_y, |T|_z\right\}$

[36]:
```
sys.specifieds = {Fx_mag: 0.0,
                  Fy_mag: 0.0,
                  Fz_mag: 0.0,
```

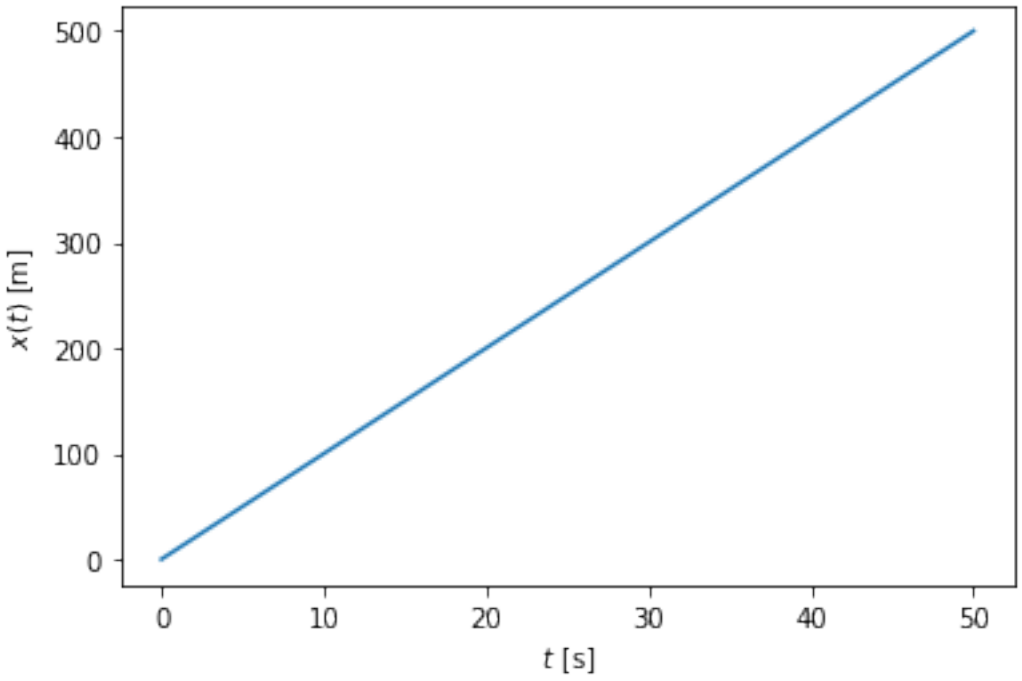
```
                    Tx_mag: 0.0,
                    Ty_mag: 0.0,
                    Tz_mag: 0.0
                   }
```

[37]:
```
states = sys.integrate()
```

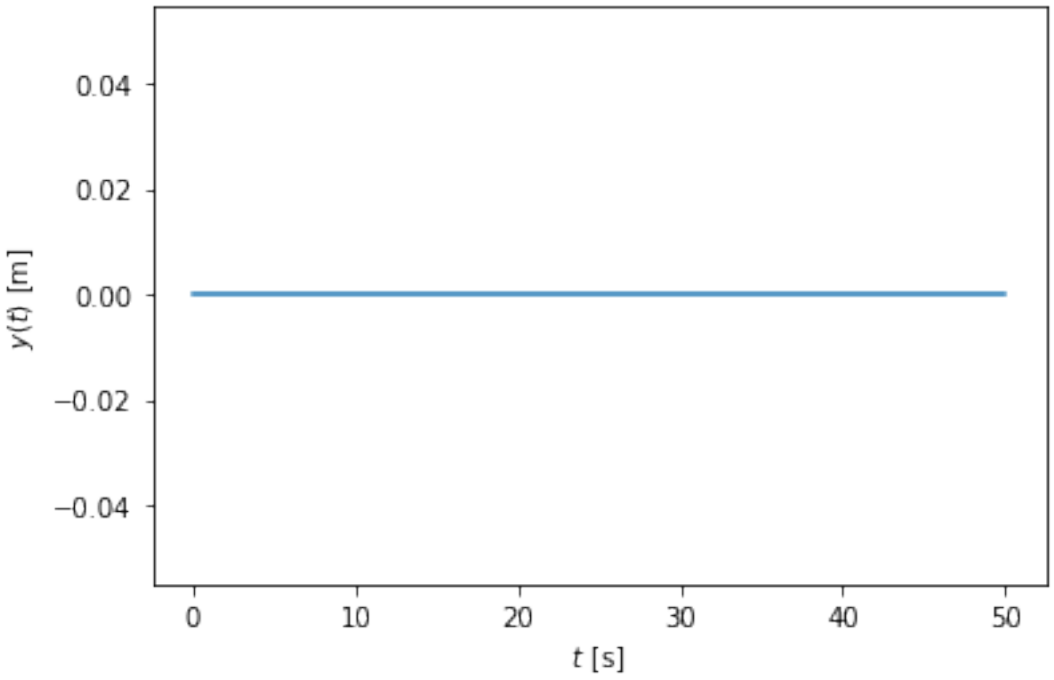
[38]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states)
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline')));
plt.show()
```



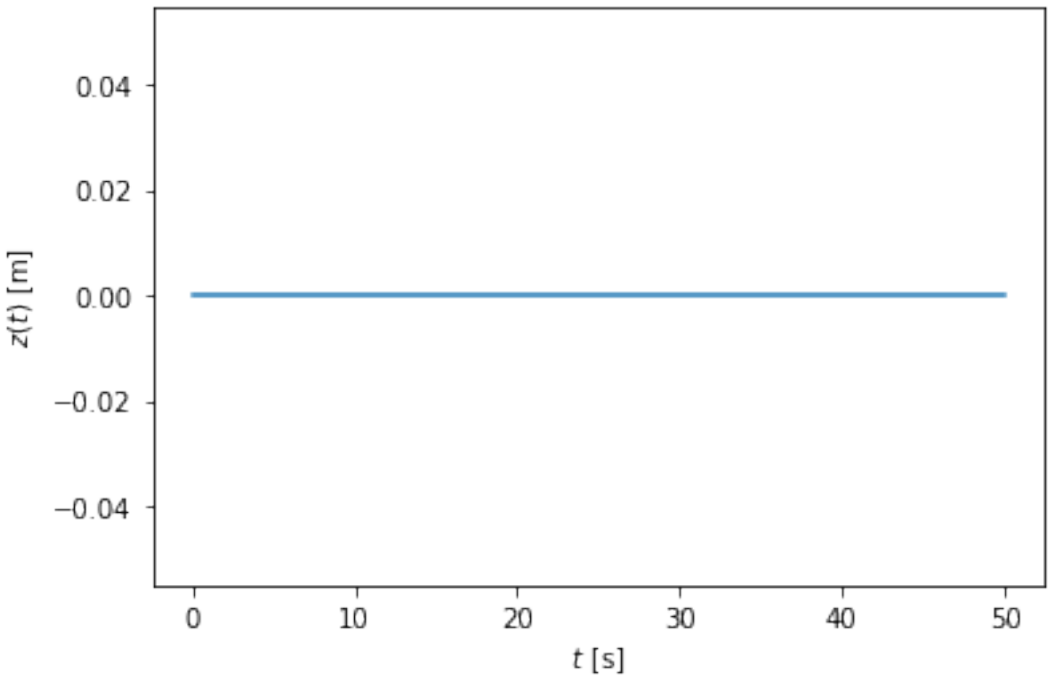
[39]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 0])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [m]'.
 ↪format(sm.latex(x, mode='inline')));
plt.show()
```



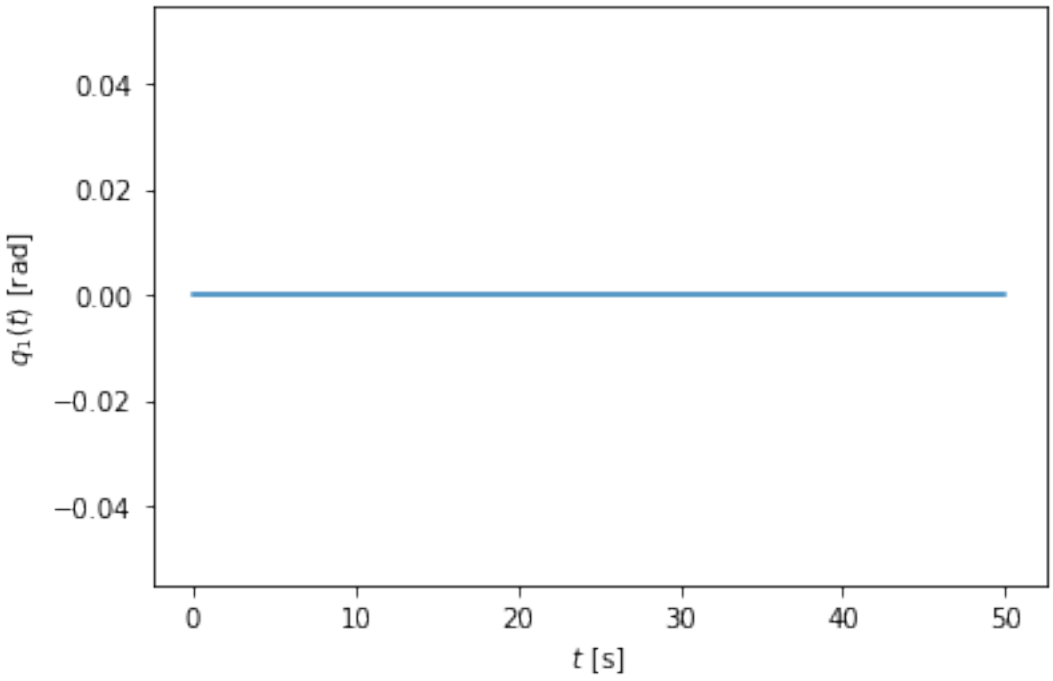
[40]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 1])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [m]'.
 ↪format(sm.latex(y, mode='inline')));
plt.show()
```



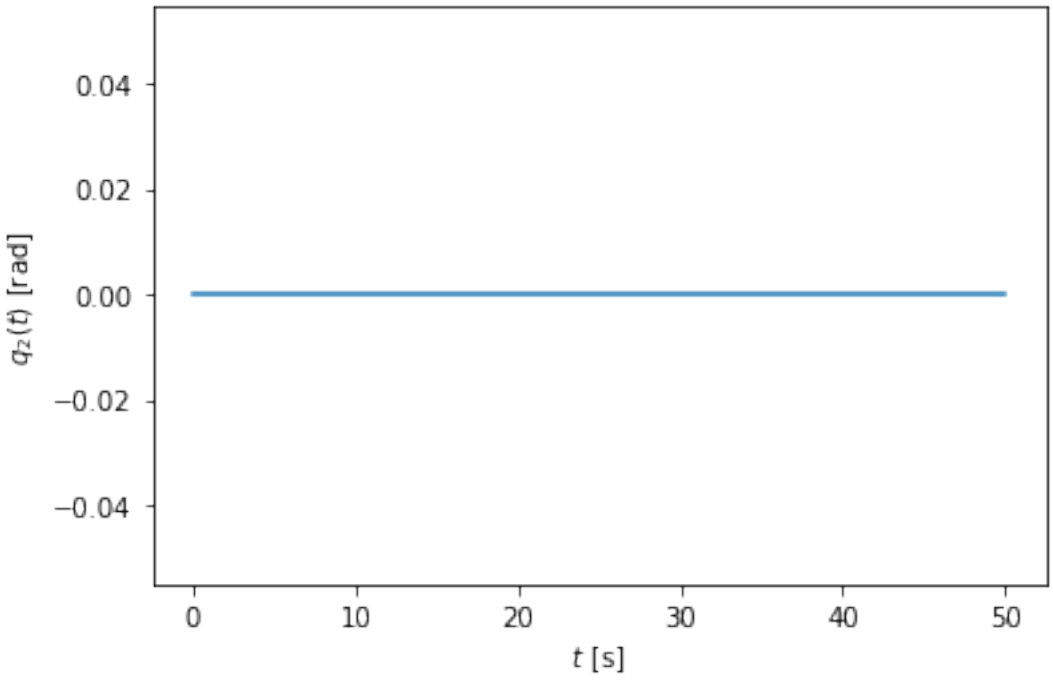
[41]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 2])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [m]'.
 ↪format(sm.latex(z, mode='inline')));
plt.show()
```



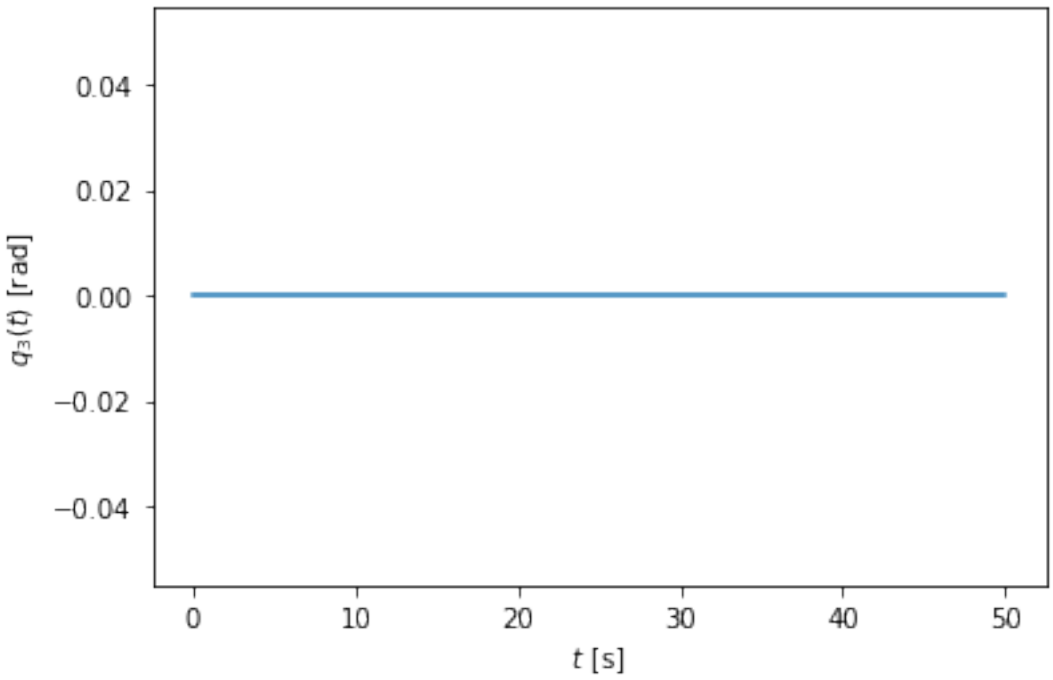
[42]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 3])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [rad]'.
 ↪format(sm.latex(q1, mode='inline')));
plt.show()
```



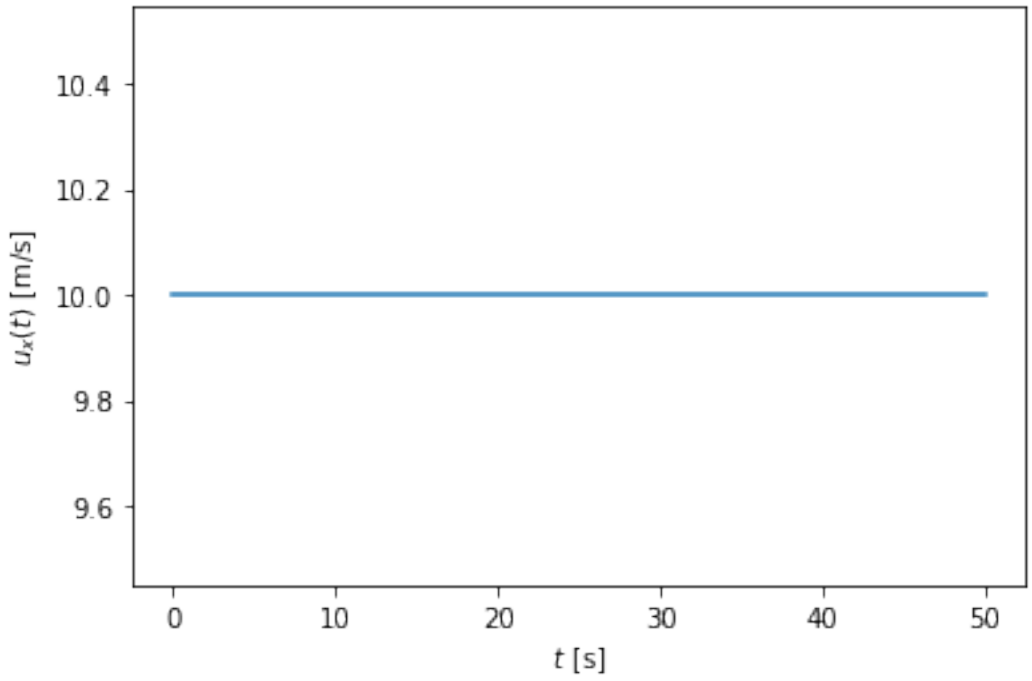
[43]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 4])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [rad]'.
 ↪format(sm.latex(q2, mode='inline')));
plt.show()
```



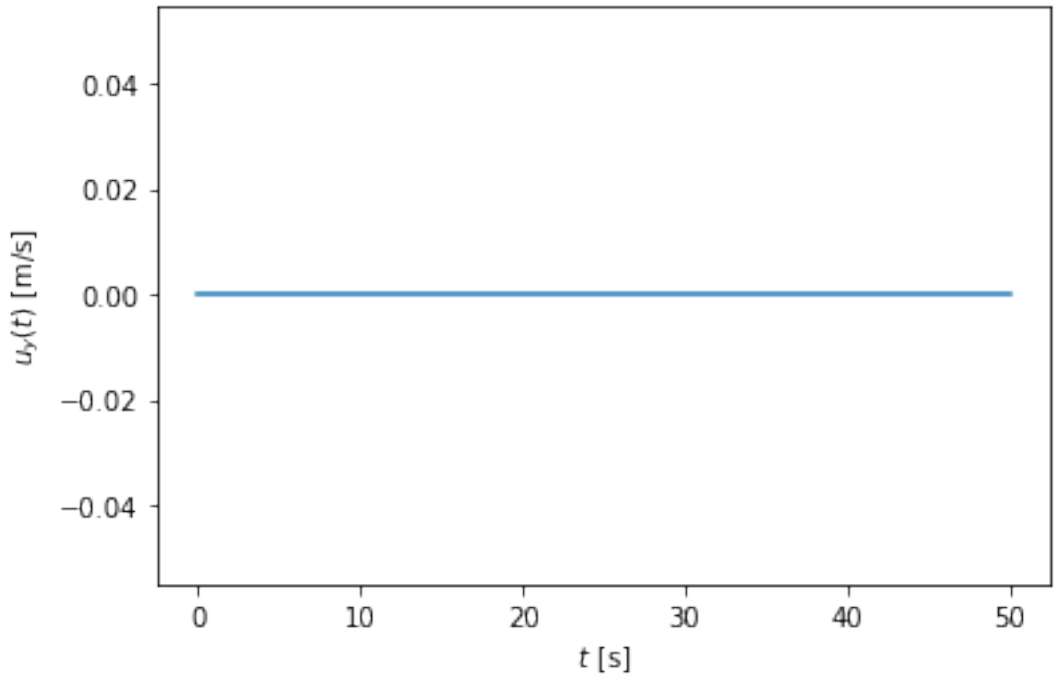
[44]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 5])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [rad]'.
 ↪format(sm.latex(q3, mode='inline')));
plt.show()
```



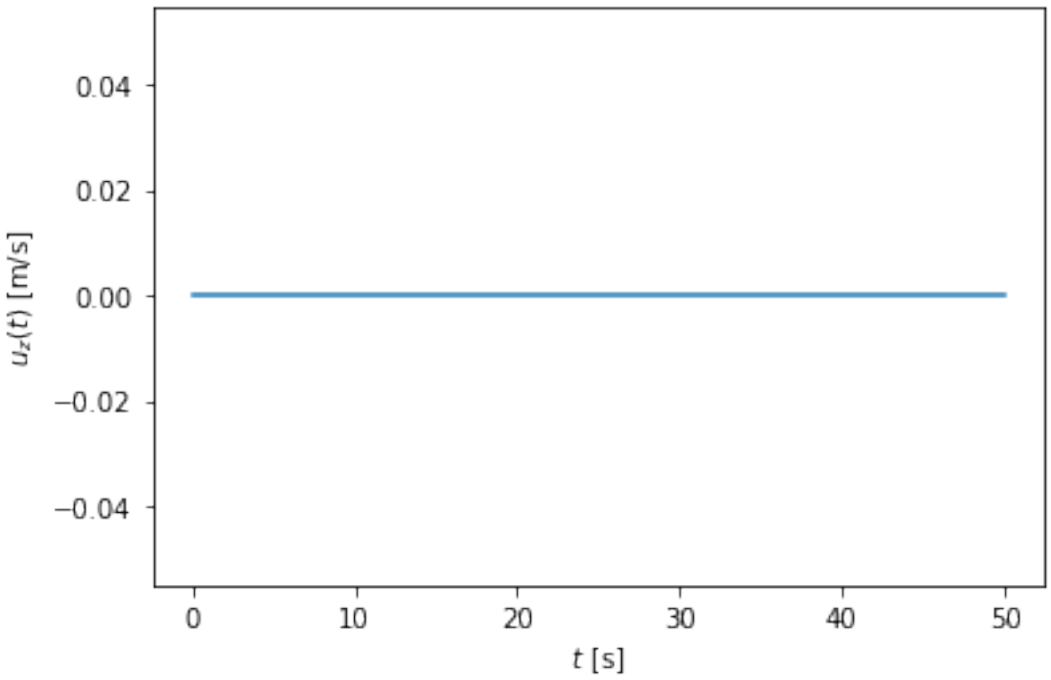
[45]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 6])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [m/s]'.
 ↪format(sm.latex(ux, mode='inline')));
plt.show()
```



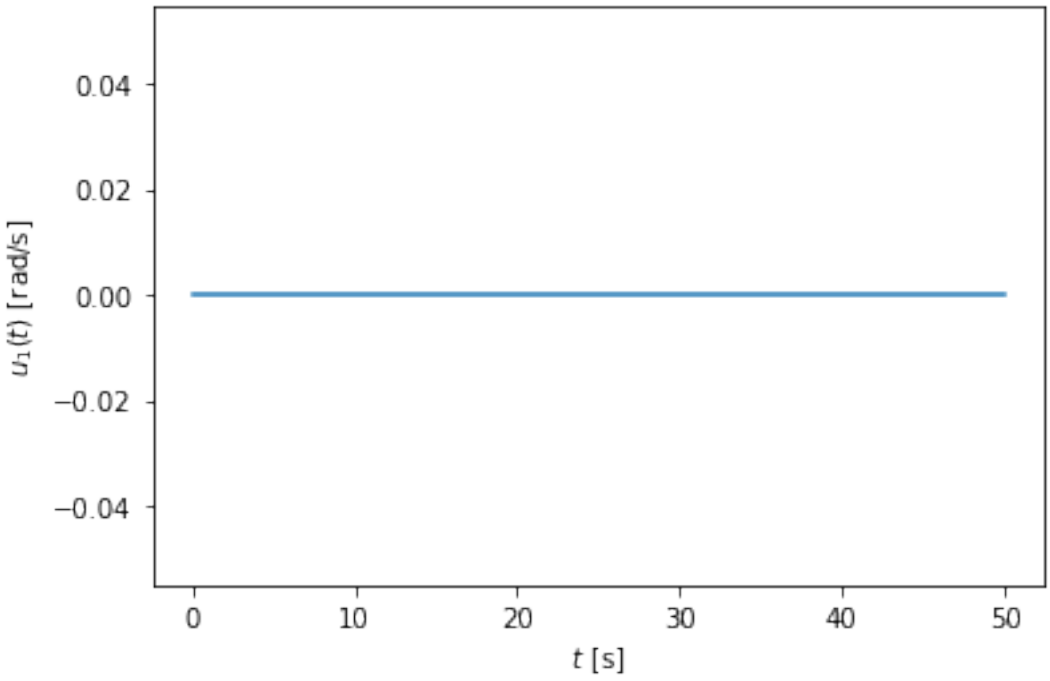
[46]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 7])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [m/s]'.
 ↪format(sm.latex(uy, mode='inline')));
plt.show()
```



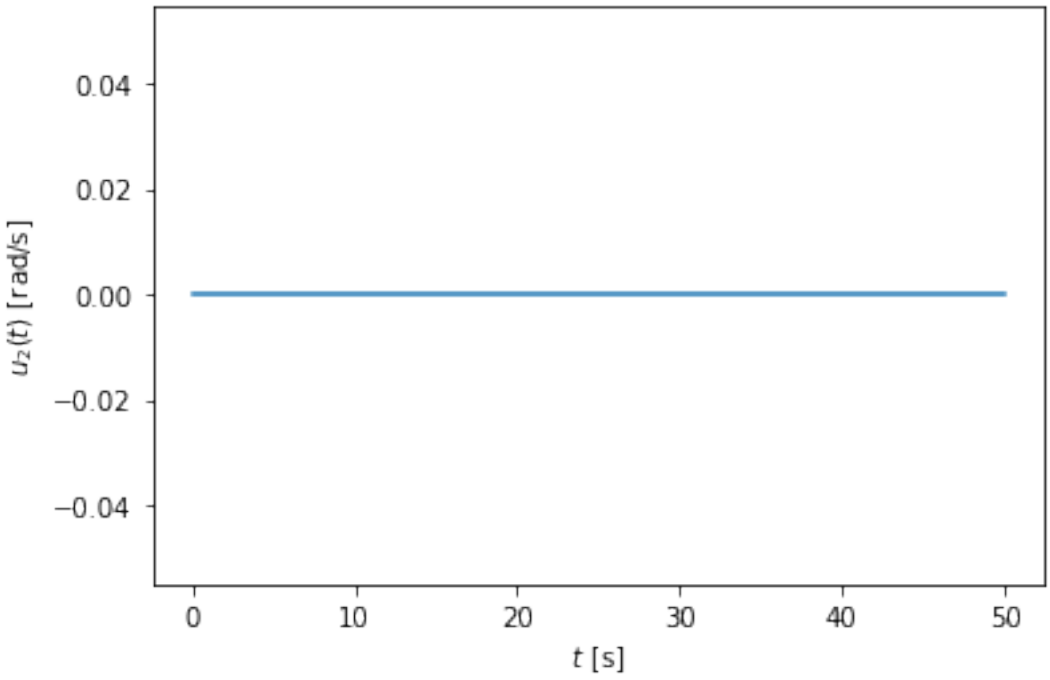
[47]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 8])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [m/s]'.
 ↪format(sm.latex(uz, mode='inline')));
plt.show()
```



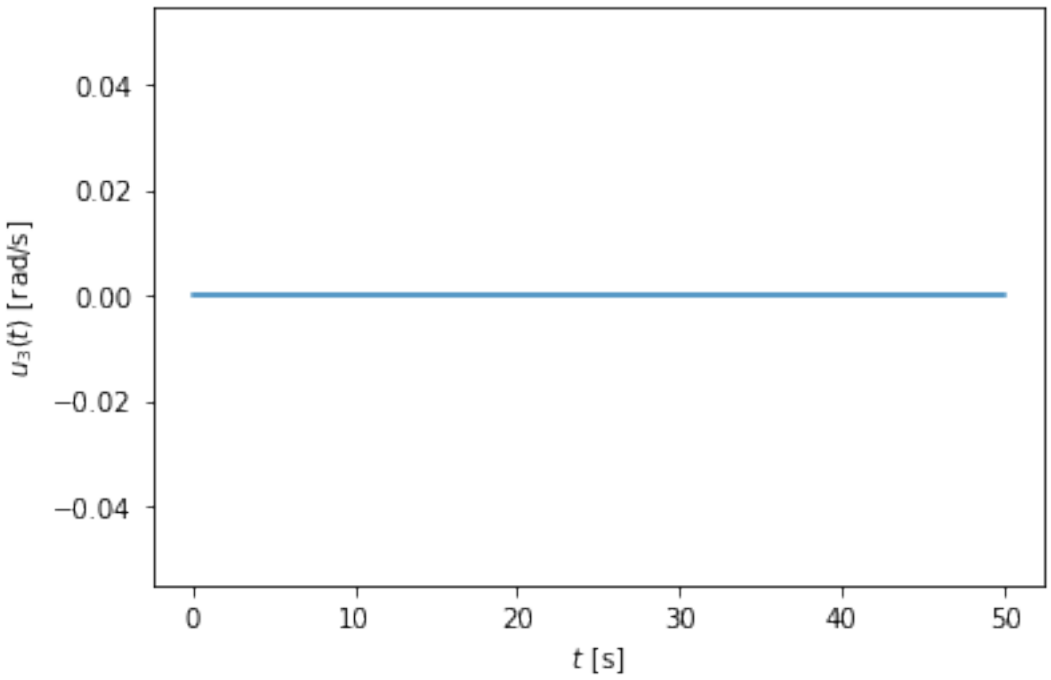
[48]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 9])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [rad/s]'.
 ↪format(sm.latex(u1, mode='inline')));
plt.show()
```



[49]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 10])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [rad/s]'.
 ↪format(sm.latex(u2, mode='inline')));
plt.show()
```



[50]:
```
fig, ax = plt.subplots()
ax.plot(sys.times, states[:, 11])
ax.set_xlabel('{} [s]'.format(sm.latex(t, mode='inline'))); ax.set_ylabel('{} [rad/s]'.
 ↪format(sm.latex(u3, mode='inline')));
plt.show()
```



0.12 3D Visualization

[51]:
```
from pydy.viz.shapes import Cube, Cylinder, Sphere, Plane
from pydy.viz.visualization_frame import VisualizationFrame
from pydy.viz import Scene
from ipywidgets import Image, Video
import pythreejs as pjs
from stl import mesh
```

[52]:
```
if configuration == "original":

    l = 0.32

    body_m_shape = Cube(l, color='black')
    body_l_shape = Cube(l, color='green')
    body_r_shape = Cube(l, color='green')

    v1 = VisualizationFrame('Body_m',
                            B,
```

```
                          C.locatenew('C_m', (1/6) * l * B.z),
                          body_m_shape)

    v2 = VisualizationFrame('Body_l',
                          B,
                          C.locatenew('C_l', (3/8) * l * -B.y),
                          body_l_shape)

    v3 = VisualizationFrame('Body_r',
                          B,
                          C.locatenew('C_l', (3/8) * l * B.y),
                          body_l_shape)

    scene = Scene(ISS, O, v1, v2, v3, system=sys)
    scene.create_static_html(overwrite=True, silent=True)

    body_m_mesh = pjs.Mesh(
        pjs.BoxBufferGeometry(l, (1/2) * l, (2/3) * l),
        pjs.MeshStandardMaterial(color='black'),
        name="Body_m"
    )

    body_l_mesh = pjs.Mesh(
        pjs.BoxBufferGeometry(l, (1/4) * l, l),
        pjs.MeshStandardMaterial(color='green'),
        name="Body_l"
    )

    body_r_mesh = pjs.Mesh(
        pjs.BoxBufferGeometry(l, (1/4) * l, l),
        pjs.MeshStandardMaterial(color='green'),
        name="Body_r"
    )

    body_m_matrices = v1.evaluate_transformation_matrix(states, list(sys.constants.
↪values()))
    body_l_matrices = v2.evaluate_transformation_matrix(states, list(sys.constants.
↪values()))
    body_r_matrices = v3.evaluate_transformation_matrix(states, list(sys.constants.
↪values()))

    body_m_track = pjs.VectorKeyframeTrack(
        name='scene/Body_m.matrix',
        times=list(sys.times),
        values=body_m_matrices)

    body_l_track = pjs.VectorKeyframeTrack(
        name='scene/Body_l.matrix',
        times=list(sys.times),
        values=body_l_matrices)

    body_r_track = pjs.VectorKeyframeTrack(
        name='scene/Body_r.matrix',
```

```
        times=list(sys.times),
        values=body_r_matrices)

    body_m_mesh.matrixAutoUpdate = False
    body_l_mesh.matrixAutoUpdate = False
    body_r_mesh.matrixAutoUpdate = False

    body_m_mesh.matrix = body_m_matrices[0]
    body_l_mesh.matrix = body_l_matrices[0]
    body_r_mesh.matrix = body_r_matrices[0]

    x_arrow = pjs.ArrowHelper(dir=[1, 0, 0], length=0.75, color='blue')
    y_arrow = pjs.ArrowHelper(dir=[0, 1, 0], length=0.75, color='red')
    z_arrow = pjs.ArrowHelper(dir=[0, 0, 1], length=0.75,color='green')

    view_width = 960
    view_height = 720

    camera = pjs.PerspectiveCamera(position=[1, 1, 1],
                                   aspect=view_width/view_height)
    key_light = pjs.DirectionalLight(position=[1, 1, 0])
    ambient_light = pjs.AmbientLight()

    scene_pjs = pjs.Scene(children=[body_m_mesh, body_l_mesh, body_r_mesh,
                                    x_arrow, y_arrow, z_arrow,
                                    camera, key_light, ambient_light])

    controller = pjs.OrbitControls(controlling=camera)
    renderer = pjs.Renderer(camera=camera, scene=scene_pjs, controls=[controller],␣
↪width=view_width, height=view_height)

elif configuration == "stowed" or "deployed":

    body_shape = Cube(0.2, color='gray')

    v1 = VisualizationFrame('Body_m',
                            B,
                            C,
                            body_shape)

    scene = Scene(ISS, O, v1, system=sys)
    scene.create_static_html(overwrite=True, silent=True)

    if configuration == "stowed":
        body_mesh = mesh.Mesh.from_file('CAD/astrobee_stowed_1.stl')
    elif configuration == "deployed":
        body_mesh = mesh.Mesh.from_file('CAD/astrobee_deployed_1.stl')

    body_vertices = pjs.BufferAttribute(array=body_mesh.vectors, normalized=False)
    body_geometry = pjs.BufferGeometry(attributes={'position': body_vertices}, )
    my_mesh = pjs.Mesh(body_geometry, pjs.MeshStandardMaterial(color='blue'),
        name='body')
```

```
    volume, cog, inertia = body_mesh.get_mass_properties()
    print("Volume                                     = {0}".format(volume))
    print("Position of the center of gravity (COG) = {0}".format(cog))
    print("Inertia matrix at expressed at the COG  = {0}".format(inertia[0,:]))
    print("                                           {0}".format(inertia[1,:]))
    print("                                           {0}".format(inertia[2,:]))

    body_matrices = v1.evaluate_transformation_matrix(states, list(sys.constants.
↪values()))

    body_track = pjs.VectorKeyframeTrack(
        name='scene/body.matrix',
        times=list(sys.times),
        values=body_matrices)

    my_mesh.matrixAutoUpdate = False

    my_mesh.matrix = body_matrices[0]

    scale = 2 * np.ndarray.max(body_mesh.points.flatten(order='C'))

    x_arrow = pjs.ArrowHelper(dir=[1, 0, 0], length=scale, color='blue')
    y_arrow = pjs.ArrowHelper(dir=[0, 1, 0], length=scale, color='red')
    z_arrow = pjs.ArrowHelper(dir=[0, 0, 1], length=scale, color='green')

    view_width = 960
    view_height = 720

    camera = pjs.PerspectiveCamera(position=[1, 1, 1],
                                   aspect=view_width/view_height)
    key_light = pjs.DirectionalLight(position=[0, 1, 1])
    ambient_light = pjs.AmbientLight()

    scene_pjs = pjs.Scene(children=[my_mesh,
                                    x_arrow, y_arrow, z_arrow,
                                    camera, key_light, ambient_light])

    controller = pjs.OrbitControls(controlling=camera)
    renderer = pjs.Renderer(camera=camera, scene=scene_pjs, controls=[controller],␣
↪width=view_width, height=view_height)
```

```
Your mesh is not closed, the mass methods will not function
            correctly on this mesh.  For more info:
            https://github.com/WoLpH/numpy-stl/issues/69

Volume                                     = 15833230.93045899
Position of the center of gravity (COG) = [-13.17222526  -0.05135002
-7.72212323]
Inertia matrix at expressed at the COG  = [ 2.73964602e+11 -2.06279823e+08
-2.05610187e+10]
                                          [-2.06279823e+08  2.89191001e+11
-9.13339047e+07]
                                          [-2.05610187e+10 -9.13339047e+07
3.29799029e+11]
```

[53]:
```
scale = body_mesh.points.flatten(order='C')
scale
```

[53]:
```
array([-345.9369  ,   -8.062765, -206.80115 , ..., -120.15808 ,
         50.419   ,  -50.8     ], dtype=float32)
```

0.13 Linearization

[54]:
```
f = fr + frstar
f
```

[54]:

$$\begin{bmatrix} -I_z \sin(q_2)\,\dot{u}_3 - \left(I_x \sin(q_3)\cos(q_2)\cos(q_3) - I_y \sin(q_3)\cos(q_2)\cos(q_3)\right)\dot{u}_2 - \left(I_x\left(-u_1 u_2 \sin(q_2)\cos(q_3) - u_1 u_3 \sin(q_3)\,c \right.\right. \end{bmatrix}$$

[55]:
```
V = {
      x: 0.0,
      y: 0.0,
      z: 0.0,
      q1: 0.0,
      q2: 0.0,
      q3: 0.0,
      ux: 0.0,
      uy: 0.0,
      uz: 0.0,
      u1: 0.0,
      u2: 0.0,
      u3: 0.0,
      Fx_mag: 0.0,
      Fy_mag: 0.0,
      Fz_mag: 0.0,
      Tx_mag: 0.0,
      Ty_mag: 0.0,
      Tz_mag: 0.0
}

V_keys = sm.Matrix([ v for v in V.keys() ])
V_values = sm.Matrix([ v for v in V.values() ])
```

[56]:
```
f_lin = f.subs(V) + f.jacobian(V_keys).subs(V)*(V_keys - V_values)
```

[57]:
```
# sm.simplify(f)
```

[58]:
```
sm.simplify(f.subs(sys.constants))
```

[58]:

$$\begin{bmatrix} \\ \\ \\ 0.5\,|T|_x \cos(q_2 - q_3) + 0.5\,|T|_x \cos(q_2 + q_3) + 0.5\,|T|_y \sin(q_2 - q_3) - 0.5\,|T|_y \sin(q_2 + q_3) + 1.0\,|T|_z \sin(q_2) - 0.01675 u_1 u_2 s \\ \\ \\ \end{bmatrix}$$

[59]:
```
us = sm.Matrix([ux, uy, uz, u1, u2, u3])
us_diff = sm.Matrix([ux.diff(), uy.diff(), uz.diff(), u1.diff(), u2.diff(), u3.diff()])
qs = sm.Matrix([x, y, z, q1, q2, q3])
rs = sm.Matrix([Fx_mag, Fy_mag, Fz_mag, Tx_mag, Ty_mag, Tz_mag])
```

If f_{lin} is used, $M_l \rightarrow$ singular

$\because$ inversion of M_l is required, use f and then substitute for V

[60]:
```
Ml = f.jacobian(us_diff).subs(sys.constants).subs(V)
Ml
```

[60]:
$$\begin{bmatrix} -16.029 & 0 & 0 & 0 & 0 & 0 \\ 0 & -16.029 & 0 & 0 & 0 & 0 \\ 0 & 0 & -16.029 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.186 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.253 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.237 \end{bmatrix}$$

[61]:
```
Cl = f.jacobian(us).subs(V)
Cl.subs(sys.constants)
```

[61]:
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

[62]:
```
Kl = f.jacobian(qs).subs(V)
sm.simplify(Kl.subs(sys.constants))
```

[62]:
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

[63]:
```
Hl = -f.jacobian(rs).subs(V)
sm.simplify(Hl.subs(sys.constants))
```

[63]:
$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

[64]:
```
A = sm.Matrix([[(-Ml.inv()*Cl), (-Ml.inv()*Kl)], [(sm.eye(6)), sm.zeros(6, 6)]])
sm.simplify(A.subs(sys.constants))
```

[64]:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

[65]:
```
A_matrix = np.array(sm.simplify(A.subs(sys.constants))).astype(np.float64)
sio.savemat('Control/Matrices/A_matrix.mat', {'A_matrix': A_matrix})
```

[66]:
```
sm.simplify(A).subs(sys.constants)*(us.col_join(qs))
```

[66]:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ u_x \\ u_y \\ u_z \\ u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

[67]:
```
B = sm.Matrix([[Ml.inv() * Hl], [sm.zeros(6, 6)]])
sm.simplify(B.subs(sys.constants))
```

[67]:

$$\begin{bmatrix} 0.0623869237007923 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0623869237007923 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0623869237007923 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5.37634408602151 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.95256916996047 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4.219409282700 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

[68]:
```
B_matrix = np.array(sm.simplify(B.subs(sys.constants))).astype(np.float64)

if configuration == "original":
    sio.savemat('Control/Matrices/B_original.mat', {'B_original': B_matrix})
```

```
elif configuration == "stowed":
    sio.savemat('Control/Matrices/B_stowed.mat', {'B_stowed': B_matrix})

elif configuration == "deployed":
    sio.savemat('Control/Matrices/B_deployed.mat', {'B_deployed': B_matrix})
```

[69]:
```
sm.simplify(B).subs(sys.constants)*(rs)
```

[69]:
$$\begin{bmatrix} 0.0623869237007923\,|F|_x \\ 0.0623869237007923\,|F|_y \\ 0.0623869237007923\,|F|_z \\ 5.37634408602151\,|T|_x \\ 3.95256916996047\,|T|_y \\ 4.21940928270042\,|T|_z \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

[70]:
```
us.col_join(qs), (sm.simplify(A).subs(sys.constants)*(us.col_join(qs)) + sm.
↪simplify(B).subs(sys.constants)*(rs)) # (x, Ax + Bu) => x_dot = Ax + Bu?
```

[70]:
$$\left(\begin{bmatrix} u_x \\ u_y \\ u_z \\ u_1 \\ u_2 \\ u_3 \\ x \\ y \\ z \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}, \begin{bmatrix} 0.0623869237007923\,|F|_x \\ 0.0623869237007923\,|F|_y \\ 0.0623869237007923\,|F|_z \\ 5.37634408602151\,|T|_x \\ 3.95256916996047\,|T|_y \\ 4.21940928270042\,|T|_z \\ u_x \\ u_y \\ u_z \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} \right)$$

[71]:
```
renderer
```

```
Renderer(camera=PerspectiveCamera(aspect=1.3333333333333333, position=(1.0, 1.0, 1.0), quaternion=(0.0,
```

[72]:
```
if configuration == "original":
    clip = pjs.AnimationClip(tracks=[body_m_track, body_l_track, body_r_track],␣
↪duration=sys.times[-1])

elif configuration == "stowed" or "deployed":
    clip = pjs.AnimationClip(tracks=[body_track], duration=sys.times[-1])

action = pjs.AnimationAction(pjs.AnimationMixer(scene_pjs), clip, scene_pjs)
action
```

```
AnimationAction(clip=AnimationClip(duration=50.0, tracks=(VectorKeyframeTrack(name='scene/body.matrix',
```