Model Predictive Control - EL2700

Finite-Time Optimal Control

Assignment 2

Division of Decision and Control Systems
School of Electrical Engineering and Computer Science
Kungliga Tekniska Högskolan

Pedro Roque, Gregorio Marchesini and Mikael Johansson

Introduction

A current trend in space robotics is focused on creating solutions to remove space debris from Earth's orbit. As humans launch more and more satellites, a considerable amount of "space garbage" is present above our atmosphere, which poses dangers to permanently inhabited facilities, such as the International Space Station, but also to new launches. The website `http://astria.tacc.utexas.edu/AstriaGraph/` sheds light into most of the tracked debris, satellites and rocket bodies currently around the Earth.

To clean all the debris, a proposed solution is to use cleaning satellites that carefully approach a debris, impelling it into the Earth atmosphere, where it will burn. The approaching maneuver is called *rendezvous*. This maneuver implies that the cleaning satellite approaches the debris and match its momentum. Once synchronized, the cleaning maneuver can be performed.

To prepare our Astrobee for such manoeuvre in space - see Figure 1 - we will first practice on the granite table to make sure that all the details work out. To this end, we will simulate a debris using Honey (an Astrobee robot), which is freely moving along a previously identified trajectory. Bumble, on the other hand, will be our cleaning robot.

For this assignment, we will use the solver MOSEK which can be used together with CVXPY. first download an academic MOSEK license at `https://www.mosek.com/license/request/?i=acp`. You should select a personal academic license. You will then receive an email over your institutional email with a file named `mosek.lic`. In order to find the license, the file `mosek.lic` should be placed in a folder called `mosek` in the home directory of your machine. Hence, on Linux/MAC create a folder called `mosek` and place it and save the `mosek.lic` inside of it as

```
$HOME/mosek/mosek.lic
```

On Windows, it will be the same

```
%USERPROFILE%\mosek\mosek.lic
```

Once the license is placed correctly you should make sure to have all the Python dependencies running the script `install_deps.py` as for the previous assignment unless you are using the virtual machine.

Figure 1: Rendezvous maneuver during the Astrobee ROAM Test Session 1 (more information on ROAM: `https://blogs.nasa.gov/stationreport/2021/05/03/`). In the image, we can see Honey (yellow robot) approaching Bumble (blue robot), having their attitudes matched. This test was a demonstration of an autonomous rendezvous manoeuvre with Model Predictive Control - something we will get to do later in the course!

Design Task

Before proceeding, let's make sure that you have all the software needed for this assignment. We recommend you to use Ubuntu 20.04 or above, but these instructions can also be completed on Windows and Mac as long as you have a working version of Python 3.6 or above. Before proceeding, install the dependencies by running the script `install_deps.py` script with Python 3.

To complete the task, carefully look into the Python files that are part of the assignment and the classes they implement. The code entry point is `task2.py`, but you will have to complete parts of the `Astrobee` and `FiniteOptimization` classes.

System Model Since in Assignment 1 we did a checkout of 1 Degree of Freedom (DoF) movement, in this assignment we will focus on the movement with 3-DoF. This means that the robot will be able to translate along two directions, and rotate along one. The state variable $\mathbf{x}$ concatenates the position and velocity along the X and Y axis, as well as the rotation angle θ and corresponding angular velocity ω around the Z-axis. Accordingly, $\mathbf{x}$ is defined as $\mathbf{x} = \begin{bmatrix} p_X & p_Y & v_X & v_Y & \theta & \omega \end{bmatrix}^T$. Moreover, the control input $\mathbf{u}$ concatenates the forces f_X and f_Y, along the X and Y axis, respectively, as well as the torque τ_Z around the Z axis, according to $\mathbf{u} = \begin{bmatrix} f_X & f_Y & \tau_Z \end{bmatrix}^T$. The linearized model, valid around a rotation $\theta \approx 0$, is given by

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_z} \end{bmatrix} \mathbf{u}(t) \tag{1}$$

where $m = 20.9$ [kg] is the mass of the Astrobee with the air-carriage and $Iz = 0.2517$ [kg m^2] is the inertia of the Astrobee and air-carriage around the Z-axis (around which the system is allowed to rotate).

Q1: Implement the model suggested in (1) in the `Astrobee` class, under the method `cartesian_ground_dynamics`.

Q2: Use the methods `c2d` and `set_discrete_dynamics` to create and define the discrete-time dynamics for Bumble.

Finite-time Optimal Control We are now ready to move to the optimization component of this assignment. In the class `FiniteOptimization`, included in the `optimization.py` file, you will find the method `__init__`, responsible for the initialization of the Finite-time solver. This solver will be used to optimize a rendezvous maneuver, so that we can approach Honey optimally. At this time, we will consider a cost function of the form $J(\mathbf{u}(t)) = \sum_{t=0}^{t=t_f} \mathbf{u}(t)^T R \mathbf{u}(t)$, where R is a diagonal, positive definite weighting matrix. We will consider the input constraints $\bar{\mathbf{u}} = -\underline{\mathbf{u}} = \begin{bmatrix} 0.85 & 0.42 & 0.04 \end{bmatrix}$, with $\bar{\mathbf{u}}$ and $\underline{\mathbf{u}}$ being the upper and lower bound of the control input. Before proceeding, make sure you understand the `__init__` method, and in a high-level, what happens in each of the 'blocks'.

Q3: In the initialization method of the solver, there is a missing component regarding the rendezvous constraints. Complete the code to perform rendezvous after the desired time. The variables involved are `x_t` and `x_ref`, for the current state and desired reference. Furthermore, the list `con_ineq` is used to store the inequality constraints of the problem. The bounds for the position and attitude are already defined in the variables `self.pos_tol` and `self.att_tol`.

Q4: Now, we will solve the finite optimization problem taking into account x_r from Honey `get_trajectory` and an initial state $\mathbf{x} = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0.01 & 0 \end{bmatrix}$ for Bumble. The function `honey.set_trajectory` will create a trajectory that spans 30 seconds. Then, with `honey.get_trajectory(t_start=25.0)` we extract the component of the trajectory that corresponds to the last 5 seconds (from 25 to 30 seconds). We will try to track these last 5 seconds of reference trajectory for Bumble to *rendezvous* with Honey. The function `solve_problem` will solve our finite-time optimization problem and return the optimal state and control input trajectories for Bumble, named `x_star` and `u_star`. Finally, we simulate the system taking into account the control inputs in `u_star`.

Extra information: in a real-life scenario, the trajectory estimate `x_ref` would be provided by an external algorithm, such as a the one in [1]. These methods are not part of this course, but they would integrate rather seamlessly with the optimization problem that you just created.

Broken Thruster During operation, Bumble suffered a problem on one of its thrusters, causing the thruster to have an erratic thrust output. The configuration `sim_env.broken_thruster()` simulates this behavior.

Q5: What do you observe when you feed the system with the previously generated control inputs? What do you believe is the root cause for this behavior?

Q6: How would you fix this problem and improve the performance of the system?

Extra Dimensions We are now on our final steps before going into orbit! To this end, we are preparing Queen to perform rendezvous using the Finite-time optimization control method that we just implemented. Now, we need to extend the model we have to an extra dimension in translation, so that Queen can move freely in this extra dimension.

Q7: Extend the model by adding a translation on Z axis, taking inspiration from the model provided in (1), such that the state for Queen is given by $\mathbf{x} = \begin{bmatrix} p_X & p_Y & p_Z & v_X & v_Y & v_Z & \theta & \omega \end{bmatrix}^T$ and the control input is $\mathbf{u} = \begin{bmatrix} f_X & f_Y & f_Z & \tau_Z \end{bmatrix}^T$. Implement this model in the function `cartesian_3d_dynamics` in the `Astrobee` class. Note that, this time, the trajectory type is set to 3D, so you should be able to simply run the simulation once you finished implementing the 3D model. Comment on the performance of the new solver `ctrl_wz` considering the time-to-solution. This information is printed just before the final cost and solver status.

To complete this design project, you should upload a zip file containing the provided code, properly completed. The task grading is based on a successful run of `task2.py`, which should provide all the results. Post all your questions on the Slack workspace.

Good Luck!

References

[1] Charles Oestreich, Antonio Teran Espinoza, Jessica Todd, Keenan Albee, and Richard Linares. On-orbit inspection of an unknown, tumbling target using nasa's astrobee robotic free-flyers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2039–2047, 2021.