Backstepping Quaternion-Based controller for a single quadrotor

Pedro Flores

July 2020

1 Some preliminaries equations to be implemented as function

There are some elements needed to develop the attitude control code, as suggestion, these mathematical operations should be developed as functions or methods in a Quaternion Class, `flyair` already has a quaternion class, you can verify if these class has these methods and if they are the same as proposed or similar, perhaps the implementation can be different but the main results should be the same. If you are not sure you can override the method.

1.1 Quaternion product

Let two quaternions defined as

$$p = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \qquad q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{1}$$

the product of two quaternions is defined as

$$pq = \begin{bmatrix} p_0q_1 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2 \\ p_0q_2 - p_1q_3 + p_2q_0 + p_3q_1 \\ p_0q_3 + p_1q_2 - p_2q_1 + p_3q_0 \end{bmatrix} \tag{2}$$

Hence, as input the function should receive two quaternions and return a quaternion.

1.2 Conjugate

Let the quaternion q as in (1), the conjugate is defined by

$$q^* = \begin{bmatrix} q_0 \\ -q_1 \\ -q_2 \\ -q_3 \end{bmatrix} \tag{3}$$

1.3 Magnitude of quaternions

Let the quaternion q as in (1), the magniude of the quaternion is given by

$$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{4}$$

1.4 Unitary quaternion and normalization

A quaternion is unitary if its magnitude is equal to 1. The controller usually are based on unitary quaternion, hence, if the quaternion is not unitary it can be normalized. The normalization of a quaternion is given by

$$q_u = \begin{bmatrix} \frac{q_0}{|q|} \\ \frac{q_1}{|q|} \\ \frac{q_2}{|q|} \\ \frac{q_3}{|q|} \end{bmatrix} \tag{5}$$

2 Backstepping attitude controller

For the attitude control the next steps have to be transformed into code.

Step 01 Normalized the quadrotor attitude quaternion, this fact is needed, since all computations are taking into account that all quaternions are unitary.

Step 02 Obtain the quaternion assciated to the angular velocity which is given by

$$\Omega = \begin{bmatrix} 0 \\ \Omega_x \\ \Omega_y \\ \Omega_y \end{bmatrix} \tag{6}$$

The variables Ω_x, Ω_y and Ω_z are the measures obtained from the IMU.

Step 03 Normalize the desired quaternion. The first approach is to verify that the controller works well, so, the first experiment might be with

$$q_d = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

Once this approach works then the desired quaternion can be changed, but it most be normalized before sending it to the controller, the normalized process is given by (4) and (5).

Step 04 Obtain the conjugate of the desired quaternion, once the desired quaternion has been normalized, the conjugated is needed, this is given by (3).

Step 05 Compute q_e. The quaternion associated to the tracking error in attitude is given by

$$q_e = q_d^* q \tag{8}$$

The quaternion product is given by (2)

Step 06 Normalized the quaternion associate to the error, this process is given by (4) and (5).

Step 07 A variable change is needed and it is defined as

$$z_1 = \begin{bmatrix} 1 - abs(q_{e0}) \\ q_{e1} \\ q_{e2} \\ q_{e3} \end{bmatrix} \tag{9}$$

where $abs(\cdot)$ is the absolute value, this function is in the `cstdlib` header source.

Step 08 Obtain the next matrix which is the result of the first feedback

$$P = 0.5 \begin{bmatrix} sign(q_{e0})q_{e1} & sign(q_{e0})q_{e2} & sign(q_{e0})q_{e3} \\ q_0 & q_3 & -q_2 \\ -q_3 & q_0 & q_1 \\ q_2 & -q_1 & q_0 \end{bmatrix} \tag{10}$$

where $sign(\cdot)$ is just the sign (positive or negative of the argument), this can be obtained as

```
if(q_e0 >= 0)
    return 1;
else
    return -1;
```

Step 09 Compute the first virtual control

$$\Omega_Q^v = -(l_1)P^\top z_1 \tag{11}$$

where l_1 is a positive constant, P is the matrix defined in (10) and z_1 is the variable change in (9).

Step 10 Extract the angular velocity vector

$$\bar{\Omega} = \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{12}$$

This values are obtained directly from the IMU.

Step 11 Compute the second tracking error which is given by

$$z_2 = \bar{\Omega} - \Omega_Q^v \tag{13}$$

Step 12 Compute the control. The control law is given by

$$\tau = S(\bar{\Omega})J\bar{\Omega} - k_2 z_2 - P^\top z_1 \tag{14}$$

J is the inertia tensor, k_2 is a positive constant, and $S(\bar{\Omega})$ is given by

$$S(\bar{\Omega}) = \begin{bmatrix} 0 & -\Omega_z & \Omega_y \\ \Omega_z & 0 & -\Omega_x \\ -\Omega_y & \Omega_x & 0 \end{bmatrix} \tag{15}$$

3 Backstepping Position controller

To control the position with the Backsteppin methodology, the next steps are needed.

Step 01 Compute the tracking error

$$\begin{aligned} e_{1x} &= x_d - x_1 \\ e_{1y} &= y_d - y_1 \\ e_{1z} &= z_d - z_1 \end{aligned} \tag{16}$$

where x_d, y_d and z_d are the references along the coordinate axis of the inertial frame, x_1, y_1 and z_1 are the quadrotor position obtained from the Optitrack

Setp 02 A virtual control is needed and is given by

$$\begin{aligned} x^v &= \alpha_{1x} e_{1x} \\ y^v &= \alpha_{1y} e_{1y} \\ z^v &= \alpha_{1z} e_{1z} \end{aligned} \tag{17}$$

where α_{1x}, α_{1y} and α_{1z} are positive constants.

Step 03 The error between the second state and the virtual controller is given by

$$\begin{aligned} e_{2x} &= x_2 - x^v \\ e_{2y} &= y_2 - y^v \\ e_{2z} &= z_2 - z^v \end{aligned} \tag{18}$$

where x_2, y_2 and z_2 are the velocities along the coordinate axis of the body frame.

Step 04 The control laws are given by

$$\begin{aligned} U_x &= -\alpha_{1x} * (e_{2x} + \alpha_{1x} * e_{1x}) - \alpha_{2x} * e_{2x} + e_{1x} \\ U_y &= -\alpha_{1y} * (e_{2y} + \alpha_{1y} * e_{1y}) - \alpha_{2y} * e_{2y} + e_{1y} \\ U_z &= -\alpha_{1z} * (e_{2z} + \alpha_{1z} * e_{1z}) - \alpha_{2z} * e_{2z} + e_{1z} + g \end{aligned} \tag{19}$$

where α_{2x}, α_{2y} and α_{2z} are positive constants.

The control vector is given by

$$\bar{U} = \begin{bmatrix} U_x \\ U_y \\ U_z \end{bmatrix} = \begin{bmatrix} -\alpha_{1x} * (e_{2x} + \alpha_{1x} * e_{1x}) - \alpha_{2x} * e_{2x} + e_{1x} \\ -\alpha_{1y} * (e_{2y} + \alpha_{1y} * e_{1y}) - \alpha_{2y} * e_{2y} + e_{1y} \\ -\alpha_{1z} * (e_{2z} + \alpha_{1z} * e_{1z}) - \alpha_{2z} * e_{2z} + e_{1z} + g \end{bmatrix} \tag{20}$$

The control input U_1 for altitude control is

$$U_1 = \frac{\sqrt{U_x^2 + U_y^2 + U_z^2}}{m} \tag{21}$$

The unitary vector associated to $\bar{U}$ is

$$\hat{\mu} = \frac{\bar{U}}{|\bar{U}|} = \frac{1}{\sqrt{U_x^2 + U_y^2 + U_z^2}} \begin{bmatrix} U_x \\ U_y \\ U_z \end{bmatrix} \tag{22}$$

For simulation purposes in Matlab the best values are $\alpha_{x1} > \alpha_{x2}$.

Defining the vector $\hat{v} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$, then the desired quaternion q_d is given by

$$\begin{aligned} q_{d0} &= \sqrt{\frac{1 + \hat{v}^\top \hat{\mu}}{2}} \\ \bar{q}_d &= \sqrt{\frac{1 + \hat{v}^\top \hat{\mu}}{2}} S(\hat{\mu}) \hat{v} \end{aligned} \tag{23}$$

where

$$S(\bar{\mu}) = \begin{bmatrix} 0 & -\mu_3 & \mu_2 \\ \mu_3 & 0 & -\mu_1 \\ -\mu_2 & a_1 & 0 \end{bmatrix} \tag{24}$$

The desired quaternion q_d is given by

$$q_d = \begin{bmatrix} q_{d0} \\ \bar{q}_d \end{bmatrix} = \begin{bmatrix} q_{d0} \\ q_{d1} \\ q_{d2} \\ q_{d3} \end{bmatrix}$$

Once q_d is computed, it can be send to the attitude control.