Chapter

Vision-Based Autonomous Control Schemes for Quadrotor Unmanned Aerial Vehicle

Archit Krishna Kamath, Vibhu Kumar Tripathi and Laxmidhar Behera

Abstract

This chapter deals with the development of vision-based sliding mode control strategies for a quadrotor system that would enable it to perform autonomous tasks such as take-off, landing and visual inspection of structures. The aim of this work is to provide a basic understanding of the quadrotor dynamical model, key concepts in image processing and a detailed description of the sliding mode control, a widely used robust non-linear control scheme. Extensive MATLAB simulations are presented to enhance the understanding of the controller on the quadrotor system subjected to bounded disturbances and uncertainties. The vision algorithms developed in this chapter would provide the necessary reference trajectory to the controller enabling it to exercise control over the system. This work also describes, in brief, the implementation of the developed control and vision algorithms on the DJI Matrice 100 to present real-time experimental data to the readers of this chapter.

Keywords: quadrotor dynamical control, unmanned aerial vehicle, vision-based control, sliding mode control

1. Introduction

In the past few years, the interest in unmanned aerial vehicle (UAV) has been growing strongly. The possibility of removing human pilots from danger as well as the size and cost of UAVs are indeed very attractive but have to be compared to the performances attained by human-piloted vehicles in terms of mission capabilities, efficiency and flexibility. The design of flight controllers able to offer to UAVs an accurate and robust control is an important step in the design of fully autonomous vehicles. In practical operations, fixed-wing UAVs have been used for years in routine surveillance missions but their lack of stationary flight capability has shifted the focus to vertical take-off and landing (VTOL) vehicles offering the possibility of being launched from virtually anywhere along with the ability to hover above a target. Several designs are available when it comes to VTOL vehicles; however, the quadrotor configuration presented in this chapter offers all the advantages of VTOL vehicles along with an increased payload capacity, a stability in hover inherent to its design (while it is the hardest flight condition to maintain for conventional helicopter) as well as an increased maneuverability [1].

In this work, the vision-based position and altitude tracking control of a quadrotor UAV is considered. This would be then on used to align the drone to the center of a pre-defined landing pad marker on which the quadrotor would autonomously land. In practical missions, the stability of the quadrotor is easily affected by abrupt changes in the input commands. The flight controller that is designed must be capable in offering an accurate and robust control to the quadrotor. The controller demonstrated in this chapter is the sliding mode controller (SMC). The sliding mode control (SMC) technique, being a non-linear control technique, has found great applications in offering robust control solutions for handling quadrotors [2–7]. This chapter will briefly describe the process of implementing a vision algorithm alongside a classical SMC for autonomous landing of the quadrotor on a stationary platform.

2. Quadrotor configuration

The quadrotor UAV is a highly non-linear, 6 DoF, Multi-Input-Multi-Output (MIMO) and under-actuated system [8]. One can describe the vehicle as having four propellers in cross configuration as shown in **Figure 1**. Quadrotor motion is controlled by varying the speed of the four rotors. A quadrotor has two sets of clockwise and two sets of counter-clockwise rotating propellers to neutralize the effective aerodynamic drag. Vertical movement of the quadrotor system is controlled by simultaneously increasing or decreasing the thrust of all rotors. Yawing motion is created by proportionally varying the speeds of counter-clockwise rotating propellers and the rolling and pitching motions are created by applying differential thrust forces on opposite rotors of the quadrotor [9].

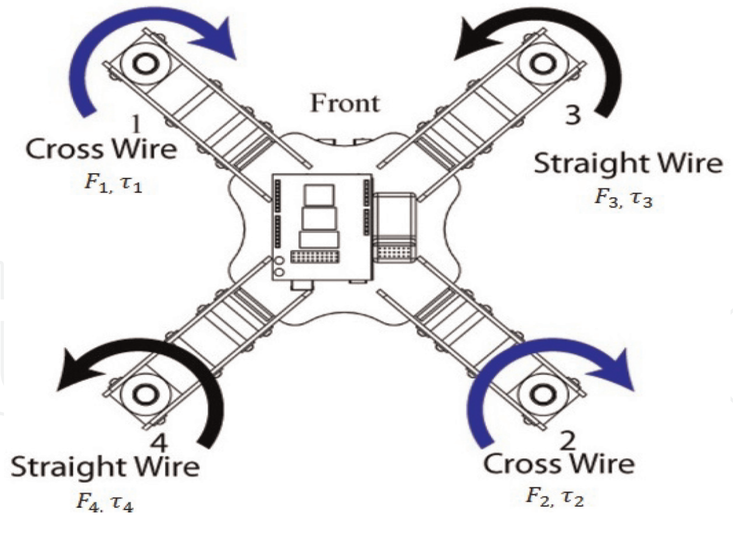


Figure 1.
Quadrotor UAV configuration.

3. Quadrotor mathematical model

The quadrotor dynamics, also called the equations of motion, are a set of 6 s order differential equations. The quadrotor being a 6 DoF plant, a total of 12 states are required to describe its motion completely. These 12 states are described using the 6 equations of motion. These play a vital role in controller design and would be extensively used in the subsequent sections of this chapter.

The kinematic and dynamic models of a quadrotor will be derived based on a Newton-Euler formalism with the following assumptions [10]:

- The quadrotor structure is assumed to be rigid and symmetrical.
- The center of gravity of the quadrotor coincides with the body fixed frame origin.
- The propellers are rigid.
- Thrust and drag are proportional to the square of propeller's speed.

The first step in developing the quadrotor kinematic model is to describe the different frames of references associated with the system. It is necessary to use these coordinate systems for the following reasons:

1. Newton's equations of motion are given the coordinate frame attached to the quadrotor.
2. Aerodynamics forces and torques are applied in the body frame.
3. On-board sensors like accelerometers and rate gyros measure information with respect to the body frame. Alternatively, GPS measures position, ground speed, and course angle with respect to the inertial frame.
4. Most mission requirements like loiter points and flying trajectories are specified in the inertial frame. In addition, map information is also given in an inertial frame.

In this case, we describe a total of frames, namely: inertial frame (F^i), the vehicle frame (F^v), the vehicle frame-1 (F^{v1}), the vehicle frame-2 (F^{v2}), and the body frame (F^b). The inertial frame is fixed at a point at ground level and uses the N-E-D notation, where N points towards north direction, E points towards east direction and D points towards earth. On the other hand, the body frame is at the center of quadrotor body, with its x axis pointing towards the front of the quadrotor, y axis pointing towards the left of the quadrotor and the z axis pointing towards the ground. The vehicle frame has an axis parallel to the inertial frame but has the origin shifted to the quadrotor's center of gravity. Vehicle frame's yaw is adjusted to match the quadrotor's yaw to get the vehicle frame-1 frame which is then pitch adjusted to get the vehicle frame-2. Finally the body frame is obtained by adjusting the roll of the vehicle frame-2.

The transformation from inertial to vehicle frame is just a simple translation. On the other hand, the transformation from vehicle to body frame is given by a rotation matrix $R_v^b(\phi,\theta,\psi)$, given by:

$$\left.\begin{aligned} &R_v^b(\phi,\theta,\psi) = R_{v2}^b(\phi)R_{v1}^{v2}(\theta)R_v^{v1}(\psi) = \\ &\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi c_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \end{aligned}\right\} \quad (1)$$

where, ϕ, θ and ψ represent the roll, pitch and yaw angles of the quadrotor measured in the vehicle frame-2, vehicle frame-1 and vehicle frame respectively. In addition to these Euler angles, the quadrotor is associated with several other state variables that describe its position, linear velocity and angular velocities. These are described as:

1. x—The inertial (north) position of the quadrotor.
2. y—The inertial (east) position of the quadrotor.
3. z—The altitude of the aircraft.
4. u—The body frame velocity in x direction in body frame.
5. v—The body frame velocity in y direction in body frame.
6. w—The body frame velocity in z direction in body frame.
7. p—The roll rate measured in body frame.
8. q—The pitch rate measured in body frame.
9. r—The yaw rate measured in body frame.

Hence a total of 12 states are used to describe the motion of the quadrotor in the 3D space.

3.1 Kinematic model

The position derivatives $(\dot{x}, \dot{y}, \dot{z})$ are inertial frame quantities and velocities (u, v, w) are in the body frame. They can be related through the transformation matrix as follows [11]:

$$\left.\begin{aligned} \frac{d}{dt}\begin{bmatrix} x \\ y \\ z \end{bmatrix} &= R_b^v \begin{bmatrix} u \\ v \\ w \end{bmatrix} \\ &= \left(R_v^b\right)^T \begin{bmatrix} u \\ v \\ w \end{bmatrix} \\ &= \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi c_\psi c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta c_\phi c_\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \end{aligned}\right\} \tag{2}$$

The relationship between absolute angles ϕ, θ, and ψ, and the angular rates p, q, and r is also complicated by the fact that these quantities are defined in different coordinate frames. The angular rates are defined in the body frame F^b, whereas the roll angle ϕ is defined in F^{v2}, the pitch angle θ is defined in F^{v1}, and the yaw angle ψ is defined in the vehicle frame F^v.

We need to relate p, q, and r to $\dot{\phi}$, $\dot{\theta}$, and $\dot{\psi}$. Since $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$ are small and noting that $R_{v1}^{v}(\dot{\psi})$, $R_{v2}^{v1}(\dot{\theta})$ and $R_{b}^{v2}(\dot{\phi})$ are all identity matrices, we get:

$$\left.\begin{aligned}\begin{bmatrix} p \\ q \\ r \end{bmatrix} &= R_b^{v2}(\dot{\phi})\begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_b^{v2}(\dot{\phi})R_{v2}^{v1}(\dot{\theta})\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_b^{v2}(\dot{\phi})R_{v2}^{v1}(\dot{\theta})R_{v1}^{v}(\dot{\psi})\begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \sin(\phi)\cos(\theta) \\ 0 & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix}\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}\end{aligned}\right\} \tag{3}$$

Inverting this, we get:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)\sec(\theta) & \cos(\phi)\sec(\theta) \end{bmatrix}\begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4}$$

3.2 Dynamic model

Let **v** be the velocity vector of the quadrotor. Newton's laws of motion hold good for inertial frames of references only. On applying these to a transnational frame, the equation modifies as follows:

$$m\frac{d\mathbf{v}}{dt_i} = f \tag{5}$$

where m is the mass of the quadrotor, f is the total applied to the quadrotor, and $\frac{d}{dt_i}$ is the time derivative in the inertial frame. From the equation of Coriolis, we have:

$$m\frac{d\mathbf{v}}{dt_i} = m\left(\frac{d\mathbf{v}}{dt_b} + \omega_{b/i} \times \mathbf{v}\right) = f \tag{6}$$

where $\omega_{b/i}$ is the angular velocity of the air-frame with respect to the inertial frame. Since the control force is computed and applied in the body coordinate system, and since ω is measured in body coordinates, we will express the above equation in body coordinates, where $\mathbf{v}^b = (u, v, w)^T$, and $\omega_{b/i}^b = (p, q, r)^T$. Therefore, in body coordinates the above equation becomes:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{1}{m}\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \tag{7}$$

where $\left[f_x f_y f_z\right]^T = f$.

For rotational motion, Newton's second law states that:

$$\frac{d\mathbf{h}^b}{dt_i} = \mathbf{m} \tag{8}$$

where **h** is the angular momentum and **m** is the applied torque. Using the equation of Coriolis we have:

$$\frac{d\mathbf{h}}{dt_i} = \left(\frac{d\mathbf{h}}{dt_b} + \omega_{b/i} \times \mathbf{h}\right) = \mathbf{m} \tag{9}$$

Again, the above equation is most easily resolved in body coordinates where $\mathbf{h}^b = \mathbf{J}\omega^b_{b/i}$, where $\mathbf{J}$ is the constant inertia matrix given by:

$$\mathbf{J} = \begin{bmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{bmatrix} \tag{10}$$

As we use a quadrotor with a symmetric frame about all three axes, $J_{xy} = J_{yz} = J_{xz} = 0$. Hence, $\mathbf{J}$ becomes:

$$\mathbf{J} = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \tag{11}$$

Defining $\mathbf{m}^b = \left[\tau_\phi \tau_\theta \tau_\psi\right]^T$ we can write Eq. (9) in the body coordinates as:

$$\mathbf{J}\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & r & -q \\ -r & 0 & p \\ p & -q & 0 \end{bmatrix}\begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix}\begin{bmatrix} p \\ q \\ r \end{bmatrix} + \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{12}$$

Hence:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} qp \end{bmatrix} + \begin{bmatrix} \frac{1}{J_x}\tau_\phi \\ \frac{1}{J_y}\tau_\theta \\ \frac{1}{J_z}\tau_\psi \end{bmatrix} \tag{13}$$

To summarize, from Eqs. (2)–(13), we obtain the 6 DoF equation of a quadrotor and is given as follows:

$$\left.\begin{aligned}
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} &= \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi c_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}\begin{bmatrix} u \\ v \\ w \end{bmatrix} \\
\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} &= \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)\sec(\theta) & \cos(\phi)\sec(\theta) \end{bmatrix}\begin{bmatrix} p \\ q \\ r \end{bmatrix} \\
\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} &= \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{1}{m}\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \\
\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} &= \begin{bmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} qp \end{bmatrix} + \begin{bmatrix} \frac{1}{J_x}\tau_\phi \\ \frac{1}{J_y}\tau_\theta \\ \frac{1}{J_z}\tau_\psi \end{bmatrix}
\end{aligned}\right\} \tag{14}$$

3.3 Forces and moments

The objective of this section is to describe the forces and torques that act on the quadrotor. Since there are no aerodynamic lifting surfaces, we will assume that the aerodynamic forces and moments are negligible. The forces and moments are primarily due to gravity and the four propellers.

As seen in **Figure 1**, each motor produces a force F and a torque τ. The total force acting on the quadrotor is given by:

$$F = F_1 + F_2 + F_3 + F_4 \tag{15}$$

The rolling torque is produced by the force difference between the motor pair 1–4 and 2–3 and is given as:

$$\tau_\phi = L(F_1 + F_4) - L(F_2 + F_3) \tag{16}$$

Similarly, the pitching torque is produced by the force difference between the motor pair 1–3 and 2–4 and is given as:

$$\tau_\theta = L(F_1 + F_3) - L(F_2 + F_4) \tag{17}$$

Due to Newton's third law, the drag of the propellers produces a yawing torque on the body of the quadrotor. The direction of the torque will be in the opposite direction of the motion of the propeller. Therefore the total yawing torque is given by:

$$\tau_\psi = \tau_1 + \tau_2 - \tau_3 - \tau_4 \tag{18}$$

The lift and drag produced by the propellers is proportional to the square of the angular velocity. We will assume that the angular velocity is directly proportional to the pulse width modulation command sent to the motor. Therefore, the force and torque of each motor can be expressed as:

$$\left.\begin{aligned} F_* &= K_1\delta_* \\ \tau_* &= K_2\delta_* \end{aligned}\right\} \tag{19}$$

where K_1 and K_2 are constants that are determined experimentally, δ_* is the motor command signal, and *—represents 1, 2, 3, and 4. Therefore, the forces and torques on the quadrotor can be written in matrix form as:

$$\begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} K_1 & K_1 & K_1 & K_1 \\ LK_1 & -LK_1 & -LK_1 & LK_1 \\ LK_1 & -LK_1 & LK_1 & -LK_1 \\ K_2 & K_2 & -K_2 & -K_2 \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{bmatrix} = M \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{bmatrix} \tag{20}$$

The control strategies derived in subsequent sections will specify forces and torques. The actual motors commands can be found as:

$$\begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{bmatrix} = M^{-1} \begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{21}$$

Note that the pulse width modulation commands are required to be between zero and one.

In addition to the force exerted by the motor, gravity also exerts a force on the quadrotor. In the vehicle frame F^v, the gravity force acting on the center of mass is given by:

$$\mathbf{f}_g^b = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{22}$$

Hence, transforming $\mathbf{f}_v^b$, we get:

$$\left.\begin{aligned} \mathbf{f}_v^b &= R_b^v \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \\ &= \begin{bmatrix} -mg\sin(\theta) \\ mg\cos(\theta)\sin(\phi) \\ mg\cos(\theta)\cos(\phi) \end{bmatrix} \end{aligned}\right\} \tag{23}$$

Therefore, transforming equation set (14), we get:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi c_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{24}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)\sec(\theta) & \cos(\phi)\sec(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{25}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \begin{bmatrix} -g\sin(\theta) \\ g\cos(\theta)\sin(\phi) \\ g\cos(\theta)\cos(\phi) \end{bmatrix} + \frac{1}{m}\begin{bmatrix} 0 \\ 0 \\ -F \end{bmatrix} \tag{26}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} qp \end{bmatrix} + \begin{bmatrix} \frac{1}{J_x}\tau_\phi \\ \frac{1}{J_y}\tau_\theta \\ \frac{1}{J_z}\tau_\psi \end{bmatrix} \tag{27}$$

Eqs. (24)–(27) represent the complete non-linear model of the quadrotor. However, they are not appropriate for control design for several reasons. The first reason is that they are too complicated to gain significant insight into the motion of the quadrotor. The second reason is that the position and orientation are relative to the inertial world fixed frame, whereas camera measurements will measure position and orientation of the target with respect to the camera frame. Hence, the above set of equations are further simplified using small angle approximation. We obtain:

$$\left.\begin{aligned}
\ddot{x} &= (\cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi))\frac{F}{m}\\
\ddot{y} &= (\cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi))\frac{F}{m}\\
\ddot{z} &= (\cos(\phi)\cos(\theta))\frac{F}{m} - g\\
\ddot{\phi} &= \frac{J_y - J_z}{J_x}\dot{\theta}\dot{\psi} + \frac{\tau_\phi}{J_x}\\
\ddot{\theta} &= \frac{J_z - J_x}{J_y}\dot{\phi}\dot{\psi} + \frac{\tau_\theta}{J_y}\\
\ddot{\psi} &= \frac{J_x - J_y}{J_z}\dot{\theta}\dot{\phi} + \frac{\tau_\psi}{J_z}
\end{aligned}\right\} \tag{28}$$

Equation set (28) would be used henceforth for developing control strategies.

4. Vision algorithm development

In order to control a system like the quadrotor, very reliable sensors are needed that can provide a good estimate of the system states. Sensors like the IMU and GPS are subjected to noise which can make them quite undesirable for control applications. Hence, an efficient method of developing control strategies for autonomous quadrotor operations is to utilize the concept of computer-vision.

Using computer vision algorithms, the on-board camera of the quadrotor can be used to confer full autonomy on the system, thereby allowing it to operate in almost any environment. This also eradicates the necessity of setting up an additional set of cameras or to calibrate the environment lighting. As long as the on-board camera is previously calibrated (just needed once) and the target to be tracked is perfectly known (marker size and ID), this system is ready to operate. The usage of ArUco markers as targets allows an easy and fast computation enabling its use in real time applications like autonomous take-off and landing.

In this chapter, let us consider the application of autonomous landing of the quadrotor on a stationary platform like a car roof-top. To enable the quadrotor to identify the landing pad, an ArUco markers board must be attached to the roof of a car. The vision algorithm must be designed to detect a specific ArUco marker ID, and provide the quadrotor's pose relative to the marker. The algorithms used for detection and identification of the marker board are reviewed in the succeeding sub-section.

4.1 The ArUco library

To detect the marker with a regular camera (RGB camera) a library called ArUco is used that was developed by Aplicaciones de la Visión Artificial (AVA) from the Universidad de Córdoba (UCO) [12]. This library is "a minimal library for Augmented Reality applications based on open source computer vision (OpenCV)" [13] and has an API for developing markers in C++ which is very useful in this work. A 100 mm Code 7 ArUco marker is shown in **Figure 2**.

A generic ArUco marker is a 2D bar-code that can be considered as a 7×7 Boolean matrix, with the outer cells filled with black (which makes a perfect square, easy to find with image processing). The remaining 5×5 matrix is a 10-bits coded ID (up to 1024 different IDs), where each line represents a couple of bits. Each line has only 2 bits of information out of the 5 bits, with the other 3 being used for error detection.

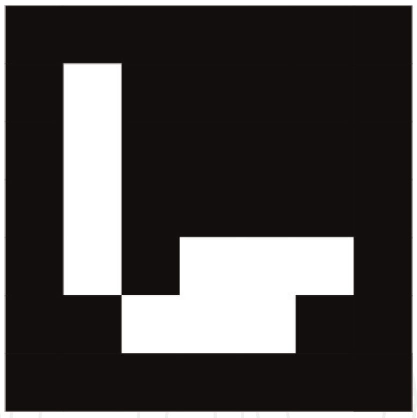


Figure 2.
ArUco marker (ID: 7, size: 100 mm).

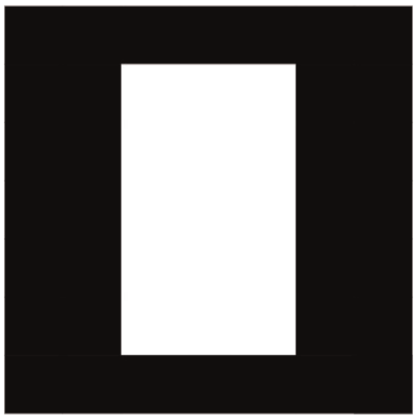


Figure 3.
ArUco marker (ID: 1023, size: 100 mm).

These extra 3 bits add asymmetry to the markers, i.e., only a few valid markers are symmetric (e.g., **Figure 3**), which allows a unique orientation detection for the markers. The codification used is a slight modification of the Hamming Code (the first bit is inverted to avoid a valid black square).

So, any ArUco marker can be created by converting a number to binary, splitting into five groups of two bits and by putting each couple in one line of the marker, from the top to the bottom. For example the marker ID of **Figure 2** is the number 7, which is (00 00 00 01 11) in binary. Using the information in **Table 1**, it can be verified that the generated marker is the same as that in **Figure 2**.

The ArUco library processes the image supplied and detects the marker ID as well as its position and orientation in the 3D world, relative to the camera. The open source code of ArUco is based in OpenCV, which is a library highly optimized for

Number	Binary	Marker Code				
0	00	⬜	⬛	⬛	⬛	⬛
1	01	⬜	⬛	⬜	⬜	⬜
2	10	⬛	⬜	⬛	⬛	⬜
3	11	⬛	⬜	⬜	⬜	⬛

Table 1.
Codification of an ArUco marker.

image processing. Therefore, all the calculations are performed in a matter of seconds so it can be used in real time applications.

The main code is not very complex and the markers detection is performed as follows:

1. Converting color image to gray image.

2. Apply adaptive shareholding.

3. Detect contours.

4. Detect rectangles:

    - Detect corners.

    - Detect linked corners.

    - Consider figures with only four connected corners.

5. For detected markers:

    - Calculate homography (from corners).

    - Threshold the area using OTSU, which assumes a bi-modal distribution and finds the threshold that maximizes the extra-class variance while keeping a low intra-class variance.

    - Detect and identify a valid marker, which respects **Table 1**, and if not detected tries the four rotations.

6. Detect extrinsic parameters (by supplying the calibration matrix, distortion matrix and physical markers dimensions).

The extrinsic parameters are calculated with the help of an OpenCV function: *solvePnP*() [14, 15]. For the marker considered, the four corners of its image and their respective 3D coordinates are provided to the algorithm, which will be:

$$\mathbf{X}_{1-4} = \begin{bmatrix} -d/2 & d/2 & d/2 & -d/2 \\ d/2 & d/2 & -d/2 & -d/2 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{29}$$

where $d[m]$ is the dimension of the side of the printed squared marker. As it can be observed, all the four points have their coordinate $Z = 0$ and their (X, Y) coordinates are disposed as a square, which means that the marker is considered to be horizontal, in the origin of the world reference, as suggested in **Figure 4** and so the extrinsic parameters will be the rotation and translation of the camera relative to the marker.

Figure 5 gives the real time implementation of the algorithm for marker detection. The vision algorithm gives the position and orientation offset of the marker center with respect to the camera center. This acts as a reference error to the controller which will use it to position the drone to the center of the ArUco marker and land it. Hence, the succeeding section of this chapter describes the control strategy development.

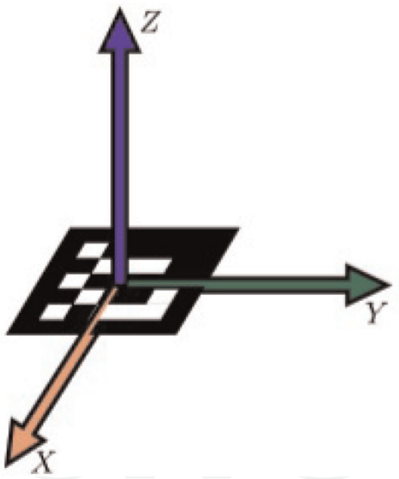
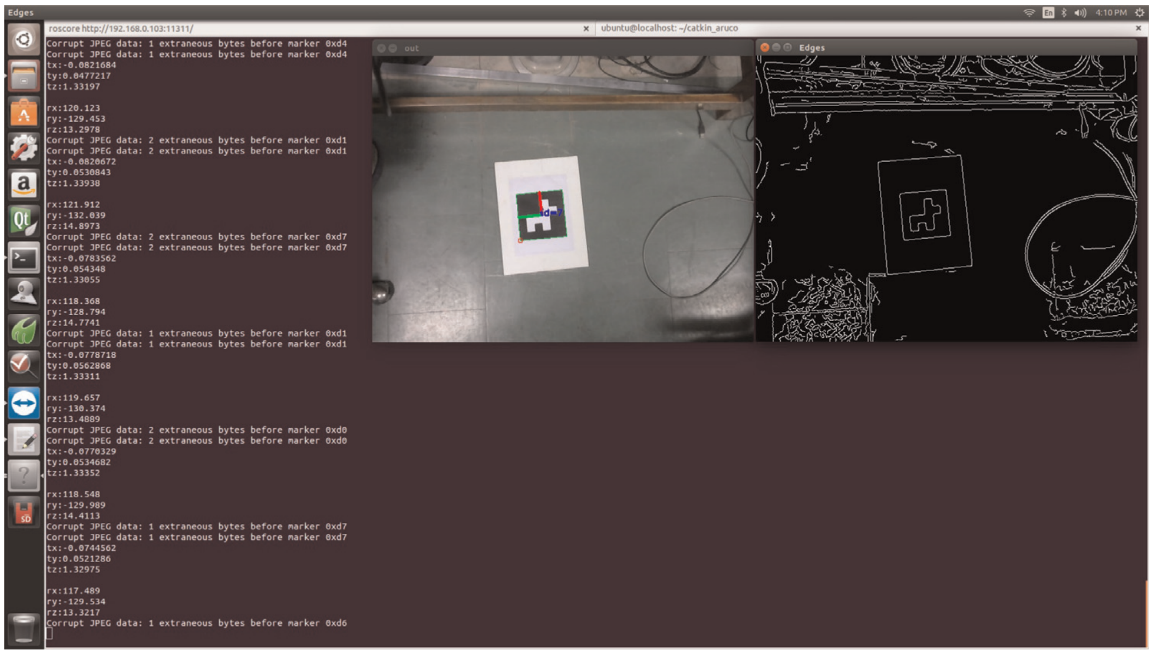


Figure 4.
A marker at the world's reference.


Edges
roscore http://192.168.0.103:11311/
ubuntu@localhost: ~/catkin_aruco
out
Edges
4:10 PM

```
Corrupt JPEG data: 1 extraneous bytes before marker 0xd4
Corrupt JPEG data: 1 extraneous bytes before marker 0xd4
tx:-0.0821684
ty:0.0477217
tz:1.33197

rx:120.123
ry:-129.453
rz:13.2978
Corrupt JPEG data: 2 extraneous bytes before marker 0xd1
Corrupt JPEG data: 2 extraneous bytes before marker 0xd1
tx:-0.0820672
ty:0.0530843
tz:1.33938

rx:121.912
ry:-132.039
rz:14.8973
Corrupt JPEG data: 2 extraneous bytes before marker 0xd7
Corrupt JPEG data: 2 extraneous bytes before marker 0xd7
tx:-0.0783562
ty:0.054348
tz:1.33055

rx:118.368
ry:-128.794
rz:14.7741
Corrupt JPEG data: 1 extraneous bytes before marker 0xd1
Corrupt JPEG data: 1 extraneous bytes before marker 0xd1
tx:-0.0778718
ty:0.0562868
tz:1.33311

rx:119.657
ry:-130.374
rz:13.4889
Corrupt JPEG data: 2 extraneous bytes before marker 0xd0
Corrupt JPEG data: 2 extraneous bytes before marker 0xd0
tx:-0.0770329
ty:0.0534682
tz:1.33352

rx:118.548
ry:-129.989
rz:14.4113
Corrupt JPEG data: 1 extraneous bytes before marker 0xd7
Corrupt JPEG data: 1 extraneous bytes before marker 0xd7
tx:-0.0744562
ty:0.0521286
tz:1.32975

rx:117.489
ry:-129.534
rz:13.3217
Corrupt JPEG data: 1 extraneous bytes before marker 0xd6
```

Figure 5.
Real-time implementation.

5. Control strategy

5.1 Introduction to sliding mode control

The sliding mode control (SMC) strategy deals with the design of a sliding manifold also called as a sliding surface which basically describe the desired behavior of the system. The designed control law works to bring the system states onto the user defined sliding surface and then slide them towards the equilibrium point along this surface. The general form of the sliding surface was proposed by Slotine and Li and is defined as [16]:

$$S = \left(\frac{d}{dt} + c_i\right)^{r-1} e = 0 \tag{30}$$

where e is the tracking error defined as $e = x - x_d$. c_i is a positive constant and r is the relative degree of the SMC. In the presence of external disturbances and uncertainties, the system trajectories may deviate from the sliding surface. This can be overcome by making the sliding surface attractive. To ensure sliding surface attractiveness, Lyapunov's theory is utilized as shown below:

Consider the Lyapunov's function V given as:

$$V = \frac{1}{2}S^2 \tag{31}$$

To make the sliding surface attractive and to guarantee asymptotic stability, $\dot{V}$ must be negative definite. In order to make $\dot{V}$ negative definite following condition must be satisfied.

$$S\dot{S} < 0 \tag{32}$$

In order to achieve finite-time convergence (global finite-time stability), the above condition is modified as:

$$S\dot{S} \le -\eta V^{1/2} \tag{33}$$

To satisfy the above inequality condition, a reaching law is selected as:

$$\dot{S} = -K\,sign(S) \tag{34}$$

where K is the gain and is always positive.

The signum function, $sign(S)$, may be defined as:

$$sign(S) = \begin{cases} +1 & S > 0 \\ 0 & S = 0 \\ -1 & S < 0 \end{cases}$$

The control law generated using SMC has two components defined as [17]:

$$u(t) = u_{eq}(t) + u_h(t) \tag{35}$$

where $u_{eq}(t)$ is the equivalent control, which can be derived by the invariance condition of the sliding surface, i.e., $S = 0$ and $\dot{S} = 0$, and $u_h(t)$ is a hitting control law also called reaching law based control, which can be obtained by testing the attractiveness condition. This hitting law is basically used to overcome the effect of uncertainties and unpredictable disturbances. Chattering appears in SMC due to signum function and can be overcome by using boundary layer method, in which the signum function is replaced by a continuous approximation function like a saturation or hyperbolic function [18].

To understand the basic steps of control law design using sliding mode, Let us consider a second order uncertain nonlinear system [19]

$$\left.\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + g(x)u(t) + d \end{aligned}\right\} \tag{36}$$

where $x = [x_1 \quad x_2]^T$ is the system state vector, $f(x)$ and $g(x) \neq 0$ are smooth nonlinear functions, and bounded uncertain term d satisfies $|d| \le d_s > 0$, and $u(t)$ is the scalar control input.

Let us define the tracking error as:

$$e = x_1 - x_d \tag{37}$$

where x_d is the desired value of the controlled variable x_1.
The sliding variable is selected as:

$$S = \dot{e} + \lambda e \tag{38}$$

where $\lambda > 0$.
Taking the time derivative of S we get:

$$\begin{aligned} \dot{S} &= \ddot{e} + \lambda \dot{e} \\ &= (\dot{x}_2 - \ddot{x}_d) + \lambda(\dot{x}_1 - \dot{x}_d) \\ &= (f(x) + g(x)u(t) + d - \ddot{x}_d) + \lambda(\dot{x}_1 - \dot{x}_d) \end{aligned} \tag{39}$$

The equivalent control effort which is designed to guarantee desired performance under nominal model is derived as the solution of $\dot{S} = 0$ without considering modeling errors and un-modeled dynamics ($d = 0$). It is represented by u_{eq} and given by:

$$u_{eq} = \frac{1}{g(x)}[\ddot{x}_d - f(x) - \lambda(x_2 - \dot{x}_d)] \tag{40}$$

The hitting control law u_h, to eliminate the effect of perturbations in conventional SMC, is chosen as:

$$u_h = -\frac{K}{g(x)} sign(S) \tag{41}$$

Hence the control law u will be the summation of u_{eq} and u_h and is written as:

$$u = \frac{1}{g(x)}[\ddot{x}_d - f(x) - \lambda(x_2 - \dot{x}_d) - K sign(S)] \tag{42}$$

Now we wish to prove that, for the system Eq. (36), with the sliding variable Eq. (38), if the control law is designed as:

$$u = \frac{1}{g(x)}[\ddot{x}_d - f(x) - \lambda(x_2 - \dot{x}_d) - K sign(S)] \tag{43}$$

with $\lambda > 0$ then the $S = 0$ will be reached in finite time. Also, the states x_1 and x_2 will converge to zero asymptotically. We use the Lyapunov's stability criteria: Let us choose the following Lyapunov candidate function as:

$$V = \frac{1}{2}S^2 \tag{44}$$

Taking the time derivative of V we get:

$$\begin{aligned} \dot{V} &= S\dot{S} \\ &= S[f(x) + g(x)u + d - \ddot{x}_d + \lambda(x_2 - \dot{x}_d)] \end{aligned} \tag{45}$$

From Eq. (42), Substitute u in Eq. (45) then

$$\begin{aligned}\dot{V} &= S(-K\,sign(S) + d)\\ &\le -K|S| + d_s|S|\\ &= -|S|(K - d_s) \le -\frac{\eta}{\sqrt{2}}|S|\end{aligned} \tag{46}$$

Since $\dot{V}$ is negative semi definite for $K \ge d_s + \frac{\eta}{\sqrt{2}}$. This ensure the finite time convergence of the sliding manifold. As a result, states are converging to desired value asymptotically. There are two phases associated with sliding mode control namely reaching phase and sliding phase. The reaching phase, is the part where the state trajectory starts from its initial condition and moves toward the sliding surface. In sliding phase, trajectories moves only on the desired sliding surface. The time taken by the states to reach sliding surface is called reaching time, denoted as t_r. To derive an expression for t_r: From Eq. (33), we can write

$$\dot{V} = -\eta V^{1/2} \tag{47}$$

Indeed, separating variables and integrating Eq. (47) over the time interval $0 \le t \le t_r$, we obtain

$$t_r = \frac{2}{\eta} V(0)^{1/2} \tag{48}$$

Therefore, a control u that is computed to satisfy Eq. (47) will drive the variable S to zero in finite time t_r and will keep it at zero thereafter. Now we extend this idea to the quadrotor by using the model represented by the equation set (28).

5.2 SMC design for quadrotor

Let us represent the non-linear model of the quadrotor as:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u} + \mathbf{d} \tag{49}$$

where:

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} \dot{\phi} \\ \left(\frac{J_y - J_z}{J_x}\right)\dot{\theta}\dot{\psi} \\ \dot{\theta} \\ \left(\frac{J_z - J_x}{J_y}\right)\dot{\phi}\dot{\psi} \\ \dot{\psi} \\ \left(\frac{J_x - J_y}{J_z}\right)\dot{\phi}\dot{\theta} \\ \dot{z} \\ -g \\ \dot{x} \\ 0 \\ \dot{y} \\ 0 \end{pmatrix}, \mathbf{d} = \begin{pmatrix} 0 \\ d_\phi \\ 0 \\ d_\theta \\ 0 \\ d_\psi \\ 0 \\ d_z \\ 0 \\ d_x \\ 0 \\ d_y \end{pmatrix} \tag{50}$$

and

$$\mathbf{g}(\mathbf{x}) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1/J_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/J_y & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/J_z \\ 0 & 0 & 0 & 0 \\ (\cos\phi\cos\theta)/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ u_x/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ u_y/m & 0 & 0 & 0 \end{pmatrix} \tag{51}$$

Here, the terms u_x and u_y are termed as virtual inputs and are evaluated as:

$$\begin{aligned} u_x &= \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ u_y &= \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \end{aligned} \tag{52}$$

From Eq. (49), the state vector can be expressed as $\mathbf{x} = \left(\phi, \dot{\phi}, \theta, \dot{\theta}, \psi, \dot{\psi}, z, \dot{z}, x, \dot{x}, y, \dot{y}\right)^T$ and the control input vector as $\mathbf{u} = (u_1, u_2, u_3, u_4)^T$ which corresponds to $\left(F, \tau_\phi, \tau_\theta, \tau_\psi\right)$. **d** represents bounded lumped disturbance which is a sum of modeling uncertainties and external wind gust disturbance associated with the quadrotor dynamics. For convenience, let the states of the system be renamed as: $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})^T$.

The quadrotor dynamical model can be split into 6 second-order sub-systems, namely the altitude, x-position, y-position, roll, pitch and yaw sub-systems. The altitude and yaw sub-systems are controlled directly by $u_!$ and u_4. However, the position sub-systems are coupled with the roll and pitch sub-systems. Hence, the concept of virtual control is utilized to develop the control scheme. Hence, u_x and u_y will control the x and y positions and u_2 and u_3 will control the roll and pitch sub-systems.

In order to design u_2, let us consider the roll subsystem which can be obtained from Eq. (49) given as:

$$\left.\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{J_y - J_z}{J_x} x_4 x_6 + \frac{u_2}{J_x} + d_\phi \end{aligned}\right\} \tag{53}$$

As mentioned previously, d_ϕ is the bounded lumped uncertainty in the roll dynamics with an upper bound of d_s. Let us consider the tracking error in roll angle as:

$$e_\phi = x_1 - \phi_d \tag{54}$$

where ϕ_d is computed from Eq. (52) as:

$$\phi_d = \sin^{-1}\left(u_x \sin\psi_d - u_y \cos\psi_d\right) \tag{55}$$

The sliding variable is defined as:

$$S_\phi = \dot{e}_\phi + \lambda_1 e_\phi \tag{56}$$

Taking the time derivative of S_ϕ:

$$\begin{aligned}\dot{S}_\phi &= \ddot{e}_\phi + \lambda_1 \dot{e}_\phi \\ &= \left(\ddot{\phi} - \ddot{\phi}_d\right) + \lambda_1\left(x_2 - \dot{\phi}_d\right)\end{aligned} \tag{57}$$

In order to eliminate the disturbance effects, the reaching law is selected as:

$$\dot{S}_\phi = -K_1 sign\left(S_\phi\right) \tag{58}$$

From Eqs. (57) and (58) the control law can be chosen as:

$$u_2 = J_x\left[\ddot{\phi}_d - \frac{J_y - J_z}{J_x}x_4x_6 - \lambda_1\left(x_2 - \dot{\phi}_d\right) - K_1 sign\left(S_\phi\right)\right] \tag{59}$$

On similar lines, u_1, u_x, u_y, u_3 and u_4 are designed as:

$$u_3 = J_y\left[\ddot{\theta}_d - \frac{J_z - J_x}{J_y}x_2x_6 - \lambda_2\left(x_4 - \dot{\theta}_d\right) - K_2 sign(S_\theta)\right] \tag{60}$$

where θ_d is computed from 52 and is given as:

$$\theta_d = \sin^{-1}\left(\frac{u_x\cos\psi_d + u_y\sin\psi_d}{\sqrt{1 - \left(u_x\sin\psi_d - u_y\cos\psi_d\right)^2}}\right) \tag{61}$$

$$u_4 = J_z\left[\ddot{\psi}_d - \frac{J_x - J_y}{J_z}x_2x_4 - \lambda_3(x_6 - \dot{\psi}_d) - K_3 sign\left(S_\psi\right)\right] \tag{62}$$

$$u_1 = \frac{m}{\cos x_1 cos x_3}\left[\ddot{z}_d + g - \lambda_4(x_8 - \dot{z}_d) - K_4 sign(S_z)\right] \tag{63}$$

The sliding variables are expressed as:

$$\left.\begin{aligned}S_\theta &= \dot{e}_\theta + \lambda_2 e_\theta \\ S_\psi &= \dot{e}_\psi + \lambda_3 e_\psi \\ S_z &= \dot{e}_z + \lambda_4 e_z\end{aligned}\right\} \tag{64}$$

With the tracking errors as:

$$\left.\begin{aligned}e_\theta &= x_3 - \theta_d; \\ e_\psi &= x_5 - \psi_d; \\ e_z &= x_7 - z_d;\end{aligned}\right\} \tag{65}$$

To achieve x and y motion control, the virtual inputs u_x and u_y are designed as:

$$\begin{aligned}u_x &= \frac{m}{u_1}\left[\ddot{x}_d - \lambda_5(x_{10} - \dot{x}_d) - K_5 sign(S_x)\right] \\ u_y &= \frac{m}{u_1}\left[\ddot{y}_d - \lambda_6\left(x_{12} - \dot{y}_d\right) - K_6 sign\left(S_y\right)\right]\end{aligned} \tag{66}$$

where

$$\left.\begin{aligned} S_x &= \dot{e}_x + \lambda_5 e_x \\ S_y &= \dot{e}_y + \lambda_6 e_y \end{aligned}\right\} \tag{67}$$

and

$$\left.\begin{aligned} e_x &= x_9 - x_d; \\ e_y &= x_{11} - y_d; \end{aligned}\right\} \tag{68}$$

As previously done, the task now is to prove that the system Eq. (49), with the sliding variables given by Eqs. (56), (64) and (67). If the control laws are designed as Eqs. (59), (60), (62), (63), and (66) then the sliding manifolds are reached in finite time t_r and the tracking error e_ϕ, e_θ, e_ψ, e_z, e_x, e_y will stay on the sliding manifolds thereafter. Consequently the controlled states $x_1, x_3, x_5, x_7, x_9, x_{11}$ will converge to the desired values in finite time t_f in the presence of bounded disturbance and uncertainties.

To do so, let us select a candidate Lyapunov function as:

$$V = \frac{1}{2}\mathbf{S}^{\mathbf{T}}\mathbf{S} \tag{69}$$

where $\mathbf{S} = \left(S_\phi, S_\theta, S_\psi, S_z, S_x, S_y\right)^T$. By taking the time derivative of the Lyapunov energy function Eq. (69), one can get:

$$\dot{V} = \mathbf{S}^{\mathbf{T}}\dot{\mathbf{S}} \tag{70}$$

where $\dot{\mathbf{S}} = \left(\dot{S}_\phi, \dot{S}_\theta, \dot{S}_\psi, \dot{S}_z, \dot{S}_x, \dot{S}_y\right)^T$. After substitution of S, $\dot{V}$ can be expressed as:

$$\dot{V} = \mathbf{S}^{\mathbf{T}}(\ddot{\mathbf{e}} + A\dot{\mathbf{e}}) \tag{71}$$

where $\dot{\mathbf{e}} = \left(\dot{e}_\phi, \dot{e}_\theta, \dot{e}_\psi, \dot{e}_z, \dot{e}_x, \dot{e}_y\right)^T$ and $\mathbf{A}$ is the diagonal matrices where $\mathbf{A} = diag\{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6\}$ with $\lambda_i > 0$. Substituting the value of designed control laws in Eq. (71):

$$\dot{V} = \mathbf{S}^{\mathbf{T}}\left(-\mathbf{Ksign}(\mathbf{S}) + \mathbf{d}\right) \tag{72}$$

where $\mathbf{K} = diag\{K_1, K_2, K_3, K_4, K_5, K_6\}$ with $K_i > 0$ and $\mathbf{d} = \left(d_\phi, d_\theta, d_\psi, d_z, d_x, d_y\right)^T$ and $sign(\mathbf{S}) = \left(sign\left(S_\phi\right), sign(S_\theta), sign\left(S_\psi\right), sign(S_z), sign(S_x), sign\left(S_y\right)\right)^T$

$$\begin{aligned} \dot{V} &= \mathbf{S}^{\mathbf{T}}\left(-\mathbf{Ksign}(\mathbf{S}) + \mathbf{d}\right) \\ &\le \sum_{i=1}^{6}(|S_i|d_{si} - K_i|S_i|) \\ &= -\sum_{i=1}^{6}|S_i|(-d_{si} + K_i) \\ &\le -\frac{\eta_1}{\sqrt{2}}\sum_{i=1}^{6}|S_i| \end{aligned} \tag{73}$$

where $|d_i| < d_{si}$. Hence, the convergence of $\mathbf{S}$ is proven by the Lyapunov stability theory. The sliding variables are converging to zero in finite time, i.e., $\mathbf{S} \to 0$. Therefore, tracking error will converge to zero asymptotically, i.e., $\mathbf{e} \to 0$.

6. Simulation results

This section presents the simulation results of the SMC described in the previous section. The tracking performance of the quadrotor is evaluated by making it track a circle of radius 1 m at an altitude of 3 m with a desired yaw angle of $\pi/6$. The tracking performance is shown in **Figures 6–9**.

The control inputs are shown in **Figures 10–13**. One can observe that there exists a presence of chattering in the control inputs when using the signum function and cannot be directly implemented in real-time hardware. To overcome this, the boundary layer approximation is utilized which smoothens the control inputs.

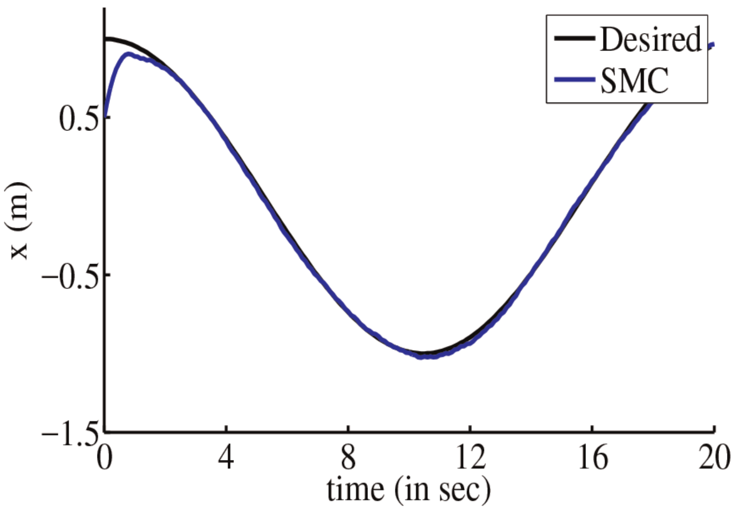


Figure 6.
X-position tracking.

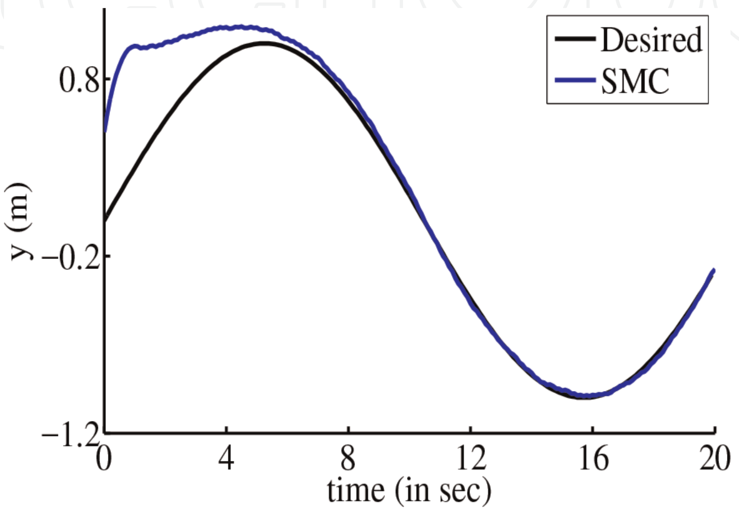


Figure 7.
Y-position tracking.

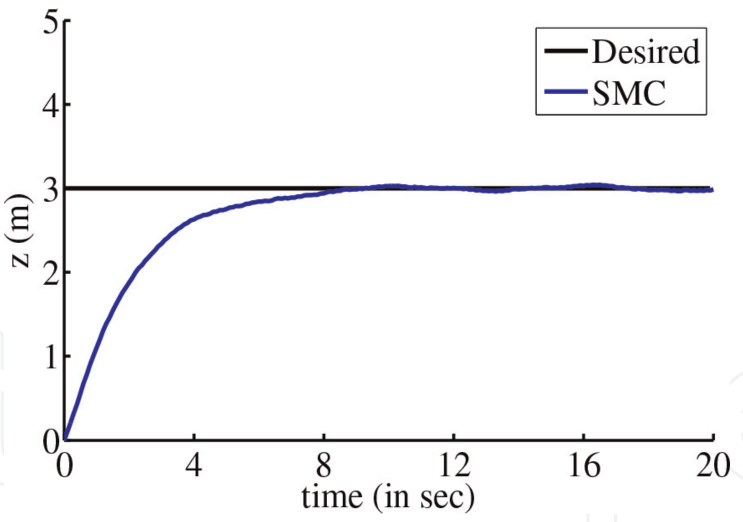


Figure 8.
Altitude tracking.

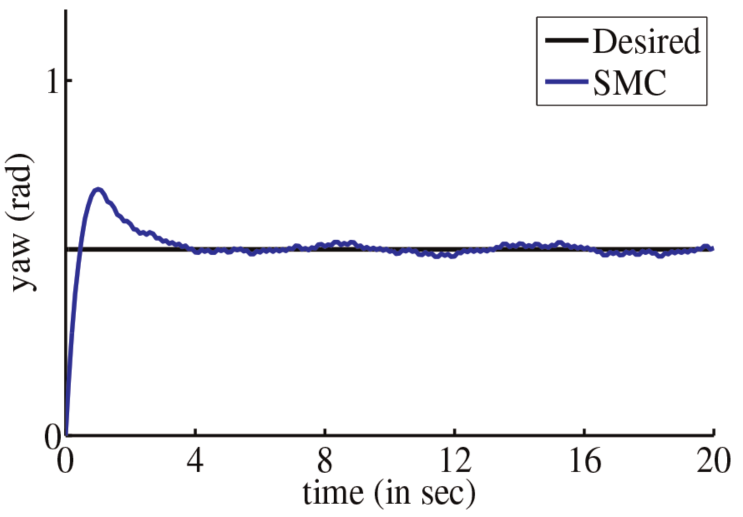


Figure 9.
Yaw tracking.

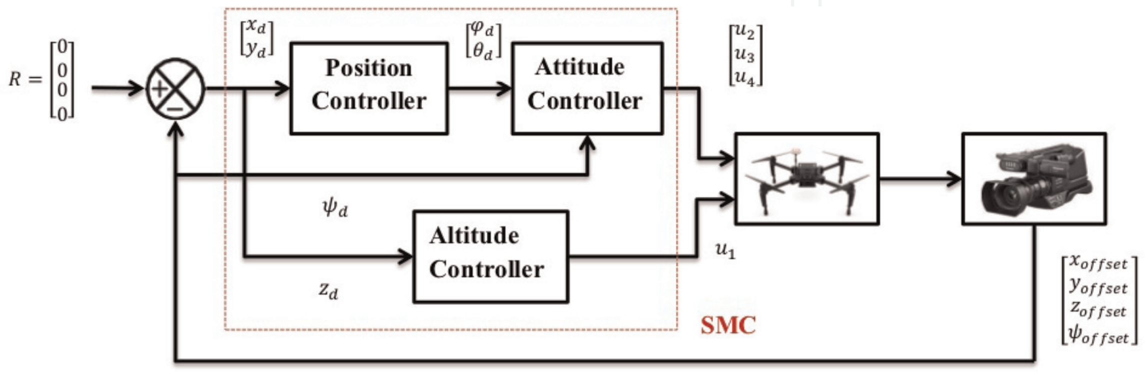

$R = \begin{bmatrix}0\\0\\0\\0\end{bmatrix}$
$\begin{bmatrix}x_d\\y_d\end{bmatrix}$
Position Controller
$\begin{bmatrix}\varphi_d\\\theta_d\end{bmatrix}$
Attitude Controller
$\begin{bmatrix}u_2\\u_3\\u_4\end{bmatrix}$
ψ_d
Altitude Controller
z_d
u_1
SMC
$\begin{bmatrix}x_{offset}\\y_{offset}\\z_{offset}\\\psi_{offset}\end{bmatrix}$

Figure 10.
u_1.

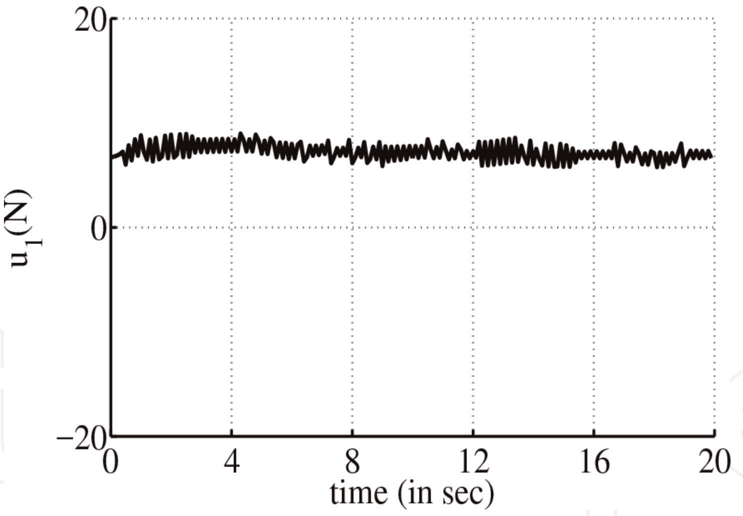


Figure 11.
u_2.

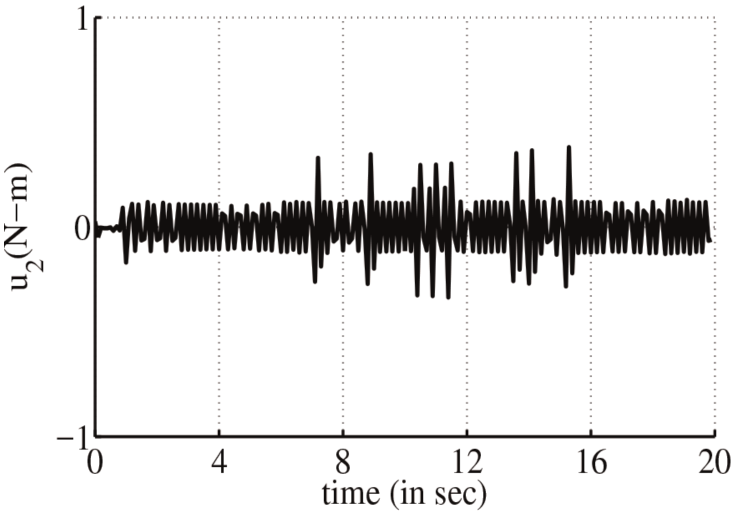


Figure 12.
u_3.

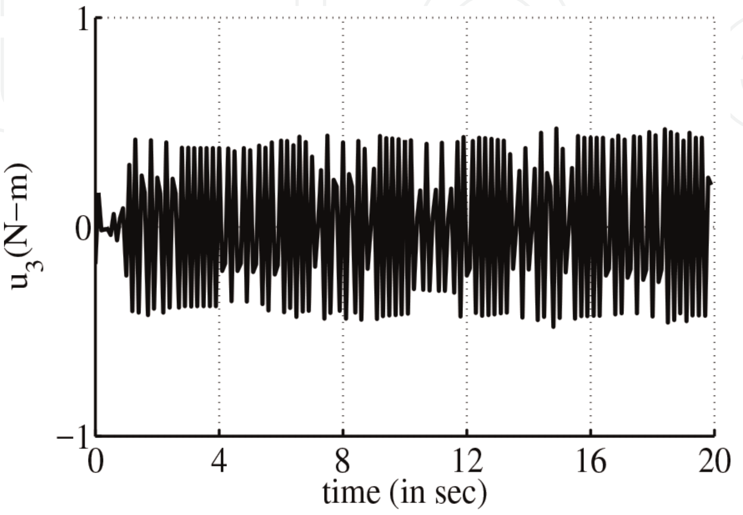


Figure 13.
u_4.

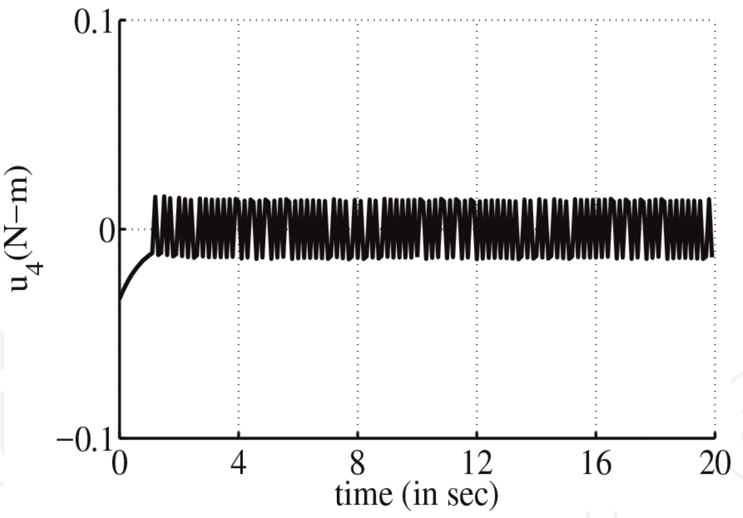


Figure 14.
Complete block diagram.

7. Vision-integrated control

Figure 14 presents an overview of the vision-integrated control. The camera captures the image and based on the vision algorithm, the position and orientation offset between the quadrotor and the marker is obtained. This offset is fed to the sliding mode controller which reduces this error and aids in landing the quadrotor at the center of the marker.

8. Hardware results

This section presents the results obtained from real-time implementation of the vision-integrated sliding mode control for the autonomous landing of the quadrotor in indoor and outdoor environments.

8.1 Hardware description

As mentioned in earlier parts of this chapter, the quadrotor used for the implementation of this work is the *DJI Matrice M100* shown in **Figure 15**. The DJI Matrice

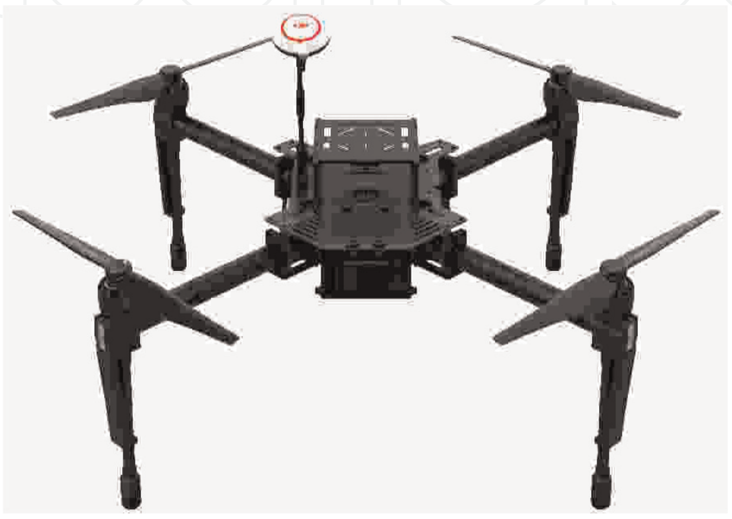


Figure 15.
DJI Matrice M100.

100 is a fully customize-able and programmable flight platform that lets its users perform operations such as pipeline health monitoring, surveillance, search and rescue and in applications requiring external sensor interface. Accompanied with the M100, a series of add-ons help in making its handling user-friendly. Similar to any other development drone in the market, the Matrice M100 comes with a programmed flight controller.

To aid in implementation of user defined controllers and task maneuvers, a separate on-board computer, named the *DJI Manifold*, is provided in **Figure 16**. The Manifold is an embedded Linux computer which incorporates a NVIDIA Tegra K1 SOC (CPU + GPU + ISP in a single chip) with both standard and extended connections and interfaces. The single GPU (Graphical Processing Unit) unit helps us run CUDA to aid in performing complex image processing operations. The Linux environment acts as a support to run ROS (Robot Operating System), which is the key element for any sorts of development on the Matrice M100. This would be mentioned in detail in the upcoming sub-section.

To gather visual data, the DJI Matrice M100 is provided with a completely controllable Zenmuse X3 Gimbal. This could be easily interfaces with the DJI Manifold for image processing. However, in this case, a separate downward facing camera is used to perform the task of vision based landing. This is done so as to keep the gimbal free to perform other tasks such as image capturing, video capturing and likewise. The downward facing camera chosen is the *LogiTech C310* camera (**Figure 17**) which can be interfaced with the manifold using an USB connection.

The landing pad is a wooden platform of dimension 4 feet $\times$ 4 feet. At the center, an AruCo marker is placed of dimension 12.5 cm $\times$ 12.5 cm. The AruCo Marker chosen is a 4 $\times$ 4 matrix of marker ID 7. The dimension of the marker is chosen such that it is clearly detected from an altitude as high as 10 m as well as from an altitude as low as 0.4 m. The landing pad setup as shown in **Figure 18** would be mounted on the roof of a car for experimental purposes.

8.2 Software description

This section briefly describes the software abstraction layer and its paradigm to control and the associated hardware flow of Matrice M100 quadrotor. As discussed in the hardware setup the DJI M100 uses DJI Manifold as its on-board computer to control and communicate with Flight controller and on-board sensors interfaced with it. *DJI On-board SDK (OSDK)* is an open source software library which enables the OBC (On-Board Computer) to handle the Input-Output data coming from the

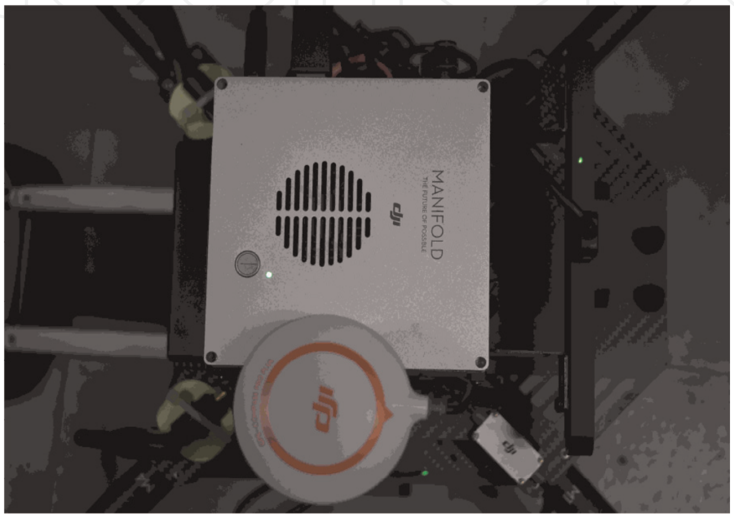


Figure 16.
DJI manifold.

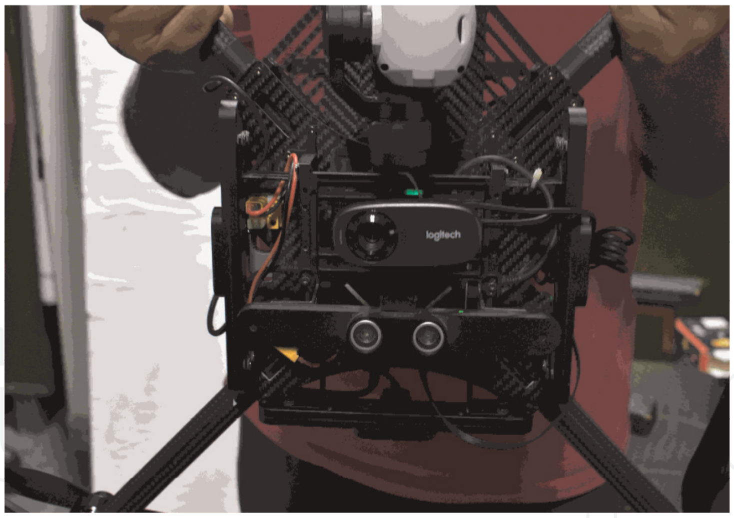


Figure 17.
LogiTech C310.

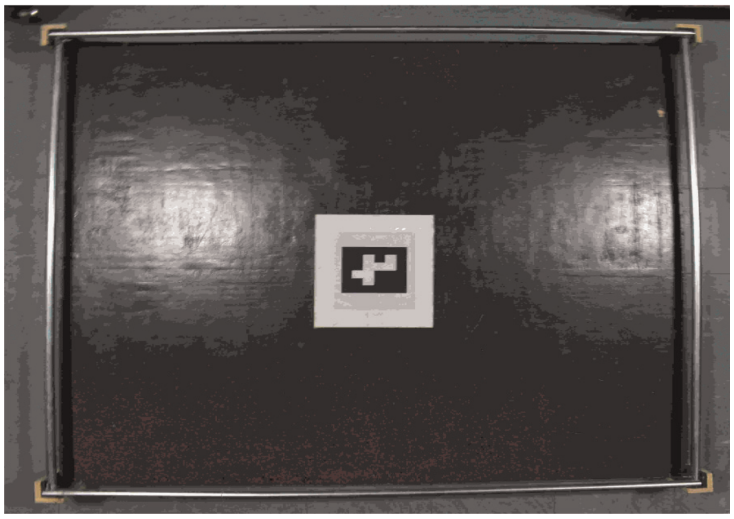


Figure 18.
Landing pad setup.

on-board control unit and sensor units. To establish the reliable network among the on-board sensor units and OBC, several serial communication protocols such as *UART1, UART2, CAN1, CAN2, USB* and *VBUS1 to VBUS5* are used. In this Paper, the main focus is on estimating the pose of the quadrotor using an on-board monocular camera connected to one of the USB ports. Other sensors, such as the *DJI Guidance*, which is connected to the *VBUS*, can be sued for fusion at different frame rates if necessary. The multi-layer hardware communication block diagram is as shown in **Figure 19**.

The multi-layer hardware connection is described in the **Figure 19**.

The on-board SDK includes:

- C++ library to access arm processor based linux(OS).
- Robot Operating System *(ROS)*: Interface and associated packages to handle multiple sensor nodes.
- *DJI Assistant2*: Real time flight simulator to verify the developed algorithms.
- *DJI OSDK API*: Used to asynchronously to send the control commands to flight control unit and s the acknowledgment from it.

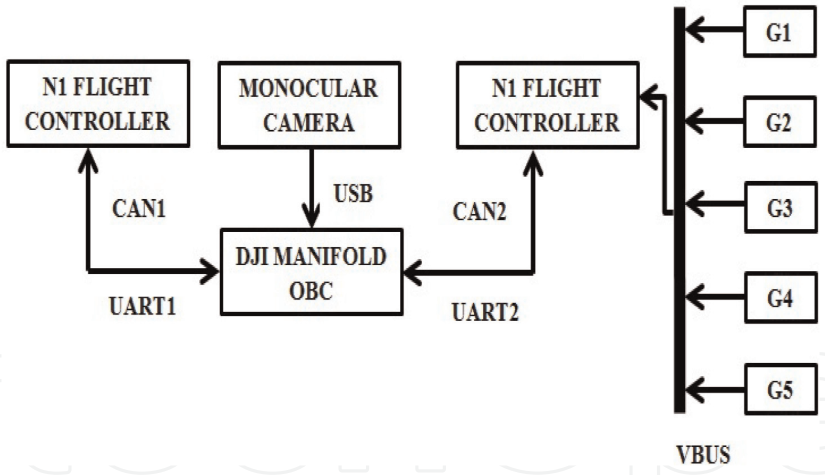


Figure 19.
OSDK architecture and software framework.

The software components of OSDK consist of APIs provided by DJI SDK library. The OSDK supports two varieties of asynchronous Programming and sends information to the OSDK workflow. The asynchronous programming mechanism works on executing the code receiving from the acknowledgement which is independent of main flow execution. The components also include:

- Serial device drivers: It communicates with flight controller and OBC via *UART*. The serial device drivers also takes care of input-output handling, memory management like locking and unlocking and interrupts.

- Thread communication: Allows inter thread communication to handle different level of signals.

- Application layer API calls: The core of on-board API is a communication between the flight control commands send from the processor to the control unit and in turn receives the acknowledgement independent of program flow. It provides callback functions. The synchronous programming API blocking calls will return only when the CMD-ACK round trip is done. This gives the assurance that the command is executed.

This process flow is depicted as shown in **Figure 20**.

8.3 Test environment description

Two test environments were used to validate the developed control algorithm. To assess the quadrotor's capability of performing vision based landing in the indoor environment, an empty plot of dimension 12 feet $\times$ 21 feet was used enclosed by nets. The plot was surrounded with obstacles on all four sides making it absolutely necessary for the drone not to move away too far away from the landing pad. The test environment is as shown in **Figure 21**. The first set of experiments were conducted using this setup. This also gave an opportunity to validate the shadow elimination that was incorporated in the drone. Note that the indoor experiment had the landing pad setup placed on the ground.

The second setup included the landing pad placed on the roof of a car as shown in **Figure 22**. It is assumed that the car is stationary when the quadrotor is performing the task of vision based landing.

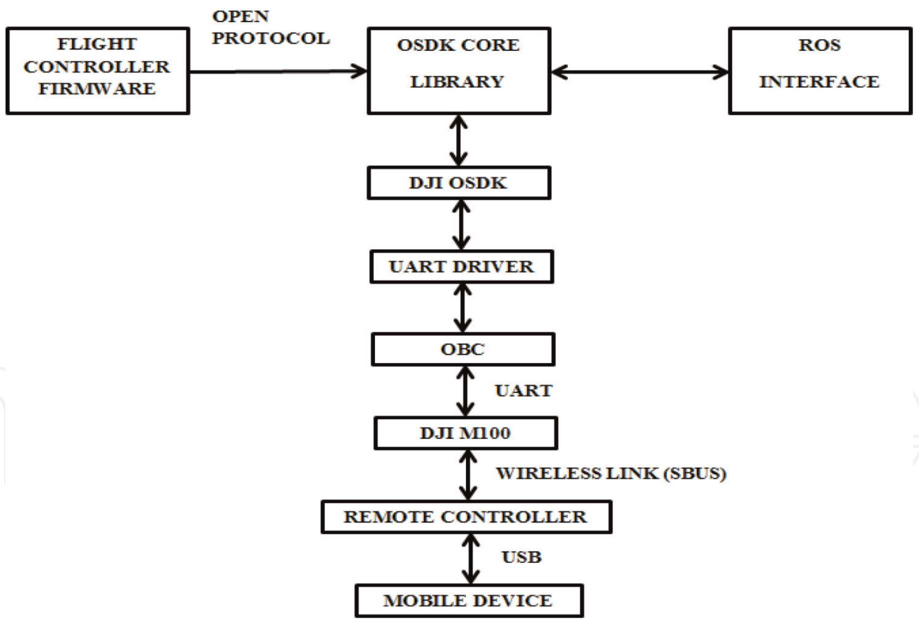


Figure 20.
OSDK software flow.

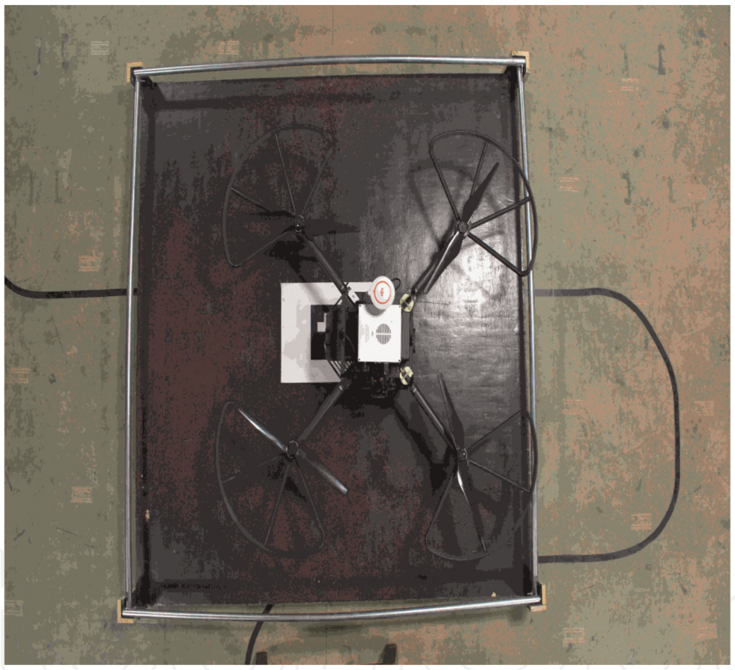


Figure 21.
Indoor test environment.

8.4 Results

8.4.1 Indoor environment

The first environment was the indoor environment with the marker placed on the ground in an enclosed space of 12 feet × 12 feet. The drone was made to autonomously lift-off and then the vision based landing node was initiated. The node simultaneously also recorded the position, velocity, acceleration and offset as the drone performed the corrections needed to align with the marker.

These results were plotted and are shown in **Figures 23–26**. Note that 1 s produces 18 samples and hence, from the time of initiation to completion, the action of

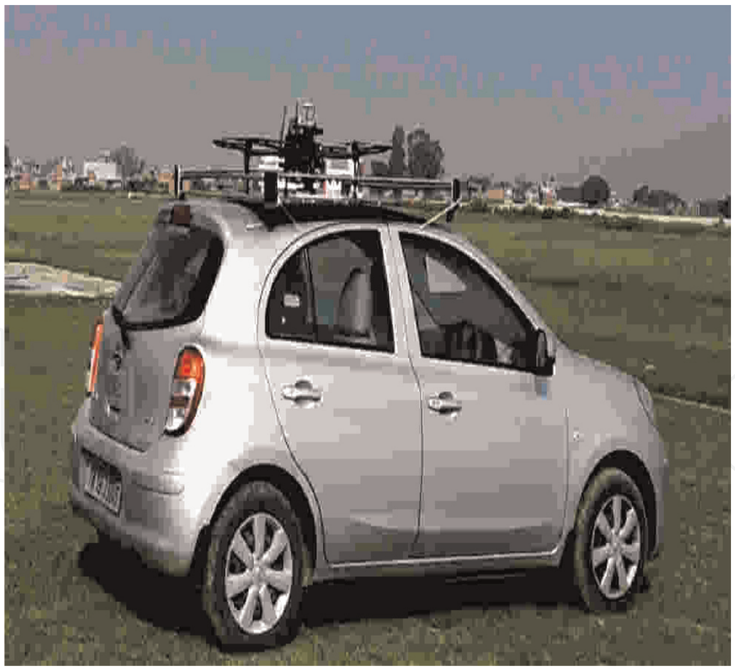


Figure 22.
Outdoor test environment.

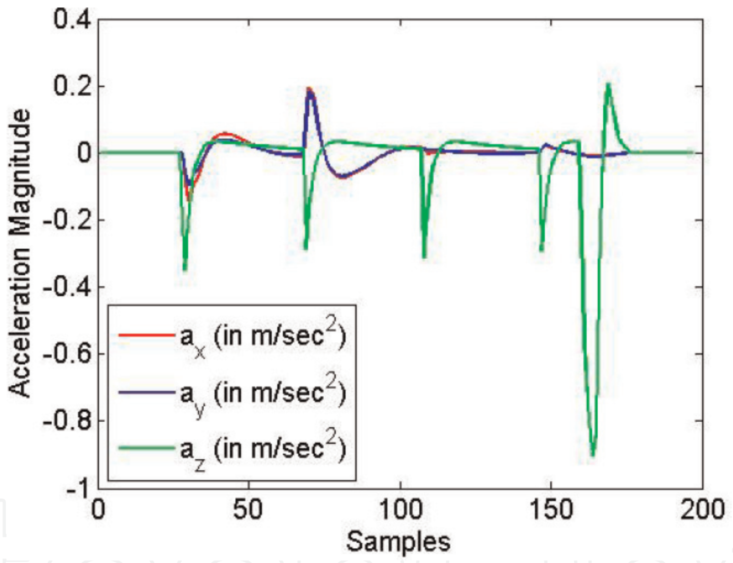


Figure 23.
Acceleration profile (Indoor Testing).

vision based landing took 30 s in this case. Over 10 trials an average error of 3.2 cm was observed with the maximum error as 6 cm from the marker center.

8.4.2 Outdoor environment

The second test environment was the outdoor environment with the landing pad mounted on the roof of a car. A 4 × 4 feet wooden board was mounted on the roof top of a car with the ArUco marker affixed to the center of this board. It was tested in an open ground with winds blowing at 10 km/hr. NW. This helped us understand the robustness of the controller designed. A slight swaying of the drone was observed, however, the designed controller managed to land the quadrotor on the marker with an average error of 4 cm with a maximum error of 7 cm over 20 trials. Once again, the acceleration, velocity, position and offset values were recorded and

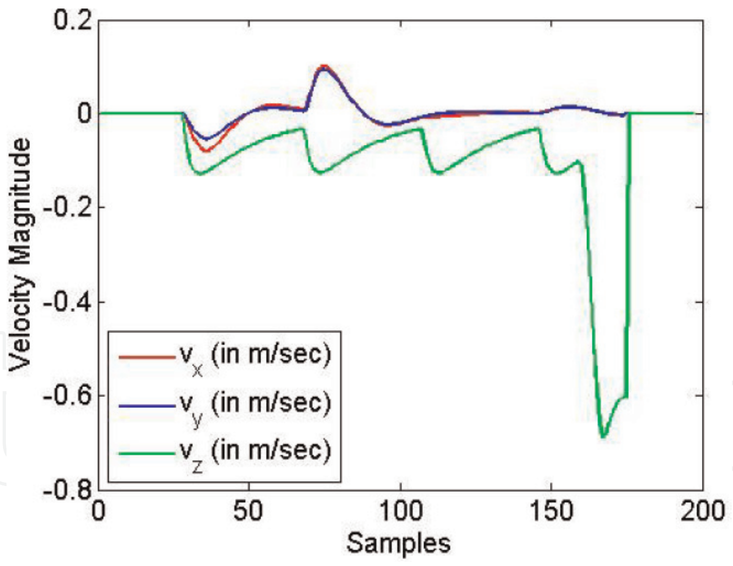


Figure 24.
Velocity profile (Indoor Testing).

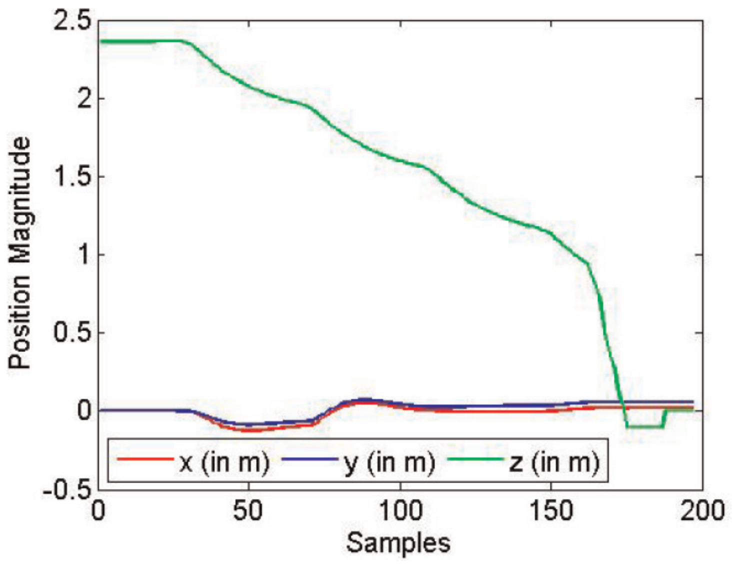


Figure 25.
Position profile (Indoor Testing).

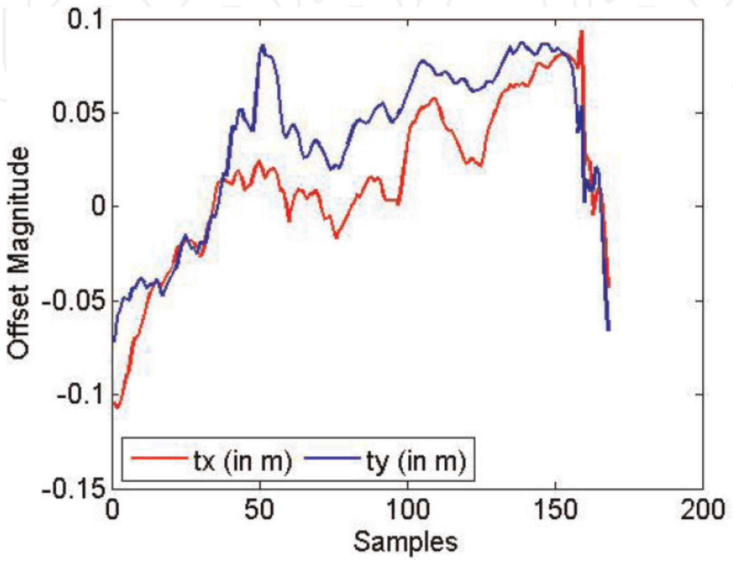


Figure 26.
Camera parameters (Indoor Testing).

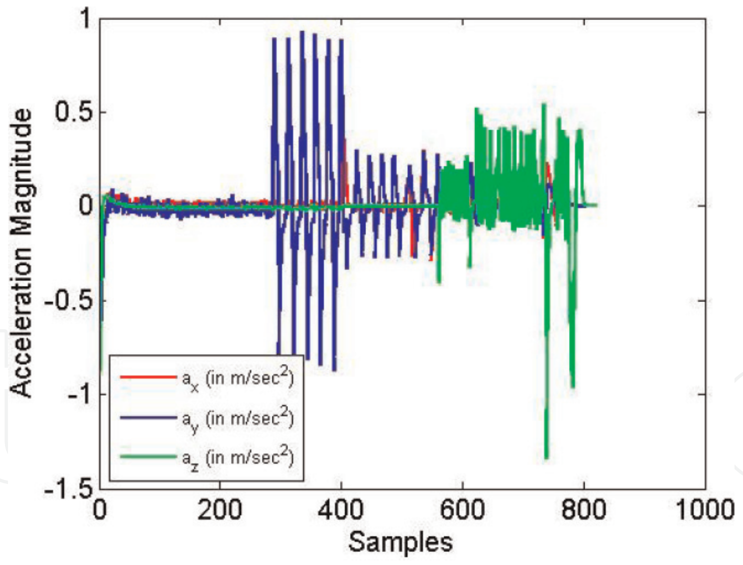


Figure 27.
Acceleration profile (Outdoor Testing).

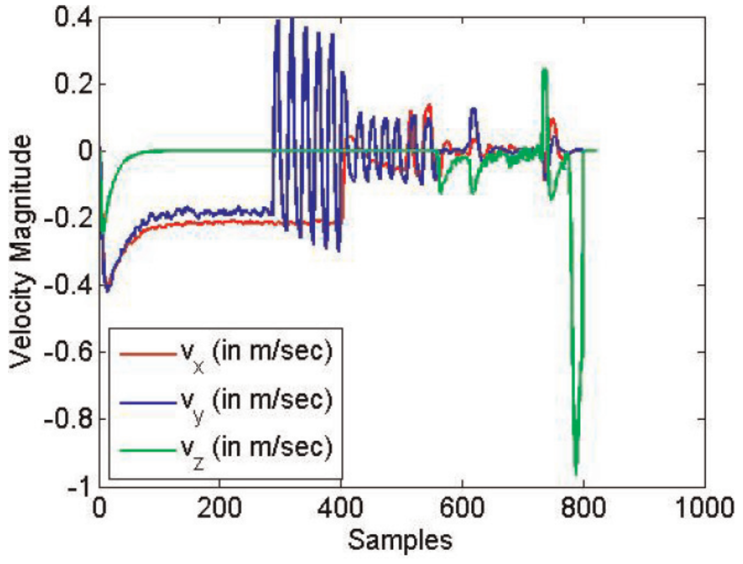


Figure 28.
Velocity profile (Outdoor Testing).

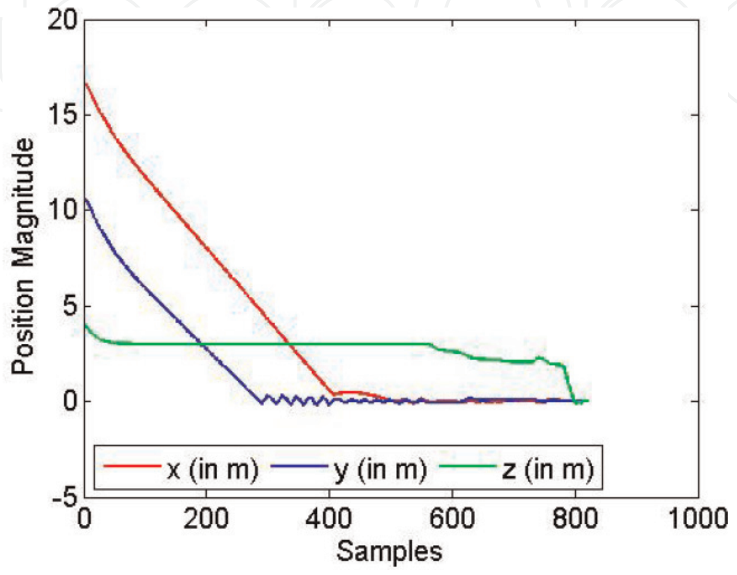


Figure 29.
Position profile (Outdoor Testing).

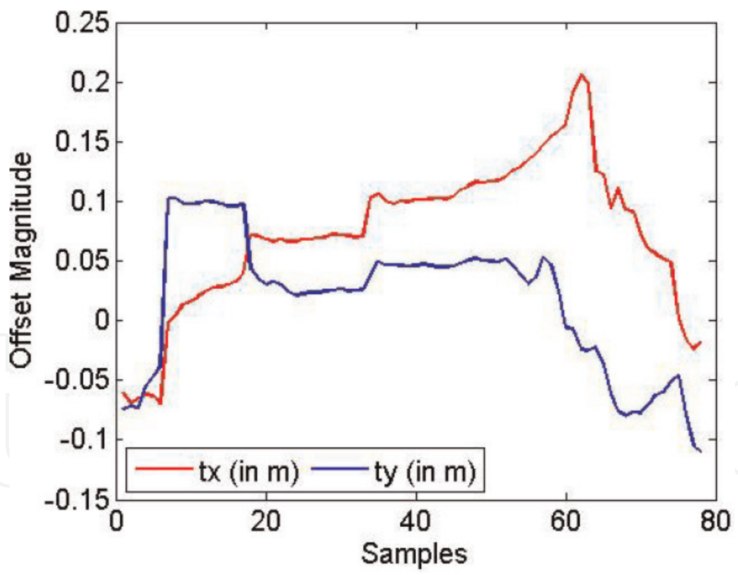


Figure 30.
Camera parameters (Outdoor Testing).

are shown in the **Figures 27–30**. The time for completion of the task from the point of initialization was found to be 43 s.

9. Conclusion

In this work, a vision-based sliding mode control for autonomous landing of a quadrotor UAV is proposed. The vision algorithm is developed to detect the centroid, position and orientation of the camera with respect to a landing pad marker (ArUco marker) placed on the roof of a car. The designed sliding mode controller proves to be effective when working alongside the developed vision algorithm and is simulated using MATLAB environment. This is then on extended to the actual experimental tests on the DJI Matrice M100, in indoor and outdoor environments. The main conclusions are summarized as follows:

1. The designed controller ensures that all the state variables converge to their reference values, even if their reference values are subjected to sudden changes.

2. The alignment of the drone over the landing pad marker is obtained by using the position and yaw offset values as inputs to the sliding mode controller.

3. The robustness of the designed controller is demonstrated among the various experimental trials in outdoor environments (subjected to winds), and the effectiveness of the proposed control scheme is also justified.

All of the results presented above are quiet promising and can be reproduced in any quadrotor system. Reference [20] demonstrates the results of the proposed work. As a future addition to this work, readers can consider using EKF to infuse IMU data with vision to enhance the tracking data. In addition, the users can also improve the proposed SMC to incorporate power rate reaching laws or super twisting laws to attenuate chattering further.

Acknowledgements

The authors would like to thank the Subhash Chand Yogi and Abhay Pratap Singh from Indian Institute of Technology-Kanpur, who have helped in running simulations and hardware experiments.

Author details

Archit Krishna Kamath, Vibhu Kumar Tripathi and Laxmidhar Behera*
Department of Electrical Engineering, Indian Institute of Technology, Kanpur, India

*Address all correspondence to: lbehera@iitk.ac.in

References

[1] Besnard L, Shtessel YB, Landrum B. Quadrotor vehicle control via sliding mode controller driven by sliding mode disturbance observer. Journal of the Franklin Institute. 2012;**349**(2):658-684

[2] Zheng EH, Xiong JJ, Luo JL. Second order sliding mode control for a quadrotor UAV. ISA Transactions. 2014; **53**(4):1350-1356

[3] Bouchoucha M, Seghour S, Tadjine M. Classical and second order sliding mode control solution to an attitude stabilization of a four rotors helicopter: From theory to experiment. In: 2011 IEEE International Conference on Mechatronics; IEEE; 2011. pp. 162-169

[4] Xu R, Özgüner Ü. Sliding mode control of a class of underactuated systems. Automatica. 2008;**44**(1): 233-241

[5] Mokhtari A, Benallegue A, Orlov Y. Exact linearization and sliding mode observer for a quadrotor unmanned aerial vehicle. International Journal of Robotics and Automation. 2006;**21**(1): 39-49

[6] Benallegue A, Mokhtari A, Fridman L. High-order sliding-mode observer for a quadrotor UAV. International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal. 2008;**18**(4-5):427-440

[7] Sharifi F, Mirzaei M, Gordon BW, Zhang Y. Fault tolerant control of a quadrotor UAV using sliding mode control. In: 2010 Conference on Control and Fault-Tolerant Systems (SysTol), IEEE; 2010. pp. 239-244

[8] Ashrafiuon H, Erwin RS. Sliding mode control of underactuated multibody systems and its application to shape change control. International Journal of Control. 2008;**81**(12): 1849-1858

[9] Tripathi VK, Behera L, Verma N. Design of sliding mode and backstepping controllers for a quadcopter. In: 2015 39th National Systems Conference (NSC), IEEE; 2015. pp. 1-6

[10] Erginer B, Altug E. Modeling and PD control of a quadrotor VTOL vehicle. In: 2007 IEEE Intelligent Vehicles Symposium, IEEE; 2007. pp. 894-899

[11] Beard R. Quadrotor dynamics and control rev 0.1

[12] Garrido-Jurado S, Muñoz-Salinas R, Madrid-Cuevas FJ, Marín-Jiménez MJ. Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition. 2014;**47** (6):2280-2292

[13] Bradski G, Kaehler A. Learning OpenCV: Computer vision with the OpenCV Library. "O'Reilly Media, Inc."; 2008

[14] Kannala J, Brandt SS. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2006;**28**(8):1335-1340

[15] Munoz-Salinas R. Aruco: A minimal library for augmented reality applications based on opencv. Universidad de Córdoba; 2012

[16] Slotine JJ, Li W. Applied Nonlinear Control. Englewood Cliffs, NJ: Prentice Hall; 1991

[17] Shtessel Y, Edwards C, Fridman L, Levant A. Sliding Mode Control and Observation. New York: Springer; 2014

[18] Runcharoon K, Srichatrapimuk V. Sliding mode control of quadrotor. In:

2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), IEEE; 2013. pp. 552-557

[19] Kuo BC, Golnaraghi F. Automatic Control Systems. Englewood Cliffs, NJ: Prentice-Hall; 1995

[20] Kamath AK, Sakthi Vignesh R, Behera L. Vision Based Autonomous Landing on Stationary Platform. [Online]. Available from: https://www.youtube.com/watch?v=d-D7enlMzlo