Mosaicing of multiplanar regions through autonomous navigation of off-the-shelf quadcopter

Meghshyam G. Prasad¹, Sona Praneeth Akula¹, Anirudh Vemula², Sharat Chandran¹

¹Department of Computer Science and Engineering, IIT Bombay, India
²Robotics Institute, Carnegie Mellon University, PA 15213, USA

E-mail: meghshyam@gmail.com

Abstract: The real world is made up of multiplanar and curved surfaces. Imaging such surfaces with an orthographic view using a handheld camera is tiring if the scene encompasses a view larger than the camera field of view. If the scene is geographically beyond the reach of humans, or access to the scene is denied, it is not possible to take such a frontal view of the scene from handheld devices. The advent of inexpensive flying vehicles such as quadcopters enables the orthographic imaging of large surfaces for later analysis either for leisure or serious scientific purposes. However, manual navigation of the quadcopter around such surfaces is tedious and results are not satisfactory. In this paper, the authors present a method for autonomous navigation of a quadcopter for imaging multiplanar surfaces. The proposed method builds on the existing parallel tracking and mapping based approach to create an approximate sparse three-dimensional map of the input scene. Next, the authors determine precise locations for obtaining videos for each bounded plane, and then autonomously maneuver the quadcopter at these waypoints. This results in an end-to-end application for visualising scenes with an off-the-shelf quadcopter.

1 Introduction

Digital imaging is omnipresent in today’s world, and there are a plethora of handheld cameras available in the market that lets us take beautiful pictures. Nevertheless, it is quite difficult to take satisfying pictures from these cameras in certain scenarios.

Consider buildings such as monuments, churches, temples, and mosques, or, for that matter, simply large structures, and the following situations:

- We are unable to access these structures proximally, or the access is restricted, or the structure is at a particularly high elevation.
- We wish to image the surfaces of these structures orthographically, i.e. by keeping the viewing direction normal to the wall.
- We wish to have fine details, and so would like to keep the camera as close to the surface as possible. As a corollary, this may require us to sweep the large surface.

In such cases, a handheld camera, such as the one available on a smartphone is less than satisfactory. On the other hand, with their flying abilities, quadcopters (quadrotor helicopters) can be used to image such surfaces from a close distance. Fig. 1a shows an art gallery set up in the foyer and due to the organiser restrictions, one cannot approach the exhibits. The details in the third poster are not clear at all.

Mosaicing: Note that, as mentioned above, we cannot capture an orthographic view of the entire large planar surface from a close distance in a single frame. In fact, as the craft flies, in all likelihood, we will get a battery of images that encompass the entire scene. These images must be stitched to get a panoramic, high-definition (HD) view. Such a view will allow us to do post-processing which may be simply to admire the picture, or, in the case of structures such as dams, to examine the structure for defects.

Unrolling: Historically, flying crafts have imaged the top view of the earth. In this work, we are interested in large upright structures; these are usually made up of multiple planes perpendicular to the earth’s surface. How should we create a panorama? For example, murals or paintings on the walls of temples are often arranged in order, possibly to depict a particular mythological story. In such cases, after the capture of these paintings spread over multiple walls by a quadcopter, we suggest an unrolled view. The entire set of images over, say, four faces of the temple, is presented on a single plane. An unrolled view of the art exhibit appears in Fig. 1b. Note that the dimensions of the original exhibit are proportionately preserved.

Path planning: If the imaging is to be done by a quadcopter, then there is further desirability that the data capture is done as quickly as possible. We examine current options available for the navigation and control of quadcopter. Most predominantly, first person view (FPV) controllers are available with current quadcopters. However, considerable manual adjustments are needed using the FPV to ensure specific depth, impacting the time of capture. Next, if we have to capture large surfaces, the manual effort is non-trivial. Finally, as always with a manual procedure (and from our experience), there is a high probability of collision during these adjustments which may cause damage to the structure.

Fig. 1 Multiplanar scene set in a foyer
(a) For aesthetic reasons, planes 1 and 2 are concave, whereas planes 2 and 3 are convex. The long-range photograph does not provide sufficient details of all planes,
(b) Desired output details are available, and the view is satisfying. The size of our output is to scale and with overall dimensions of 3427 × 863 pixels
being imaged (perhaps these are invaluable, historic monuments!). Hence, there is a requirement of developing a stable software technique for autonomous navigation and control of quadcopters in indoor as well as outdoor scenarios. Expense: There are various quadcopters available in the market ranging from, say, USD 150 to say USD 15,000. The more expensive the machine, the more features one may expect to have. Our goal in this work is to present a compact algorithm and implementation [https://github.com/meghshyam/multiplanar.] that can be used with, what may be considered as, the most basic quadcopter. This demands a sophisticated algorithm and robust implementation. We show our solution on an off-the-shelf quadcopter (Parrot AR.Drone, current price as of writing USD 150). This quadcopter can be flown indoor as well as outdoor (meets FAA requirements), weighs 4 lbs, does not contain a Global Positioning System (GPS), has a frontal camera of 720 p and 30 fps, with a flight time of ~15 min.

There could be some misgiving about the errors such an inexpensive machine can cause in the rendering. However, as noted elsewhere, we succeed in maintaining exact proportional ratios of the imaged surfaces.

1.1 Problem definition

The problems to which we provide a solution in this paper are

• Enable orthographic, fine-detailed imaging of large upright multiplanar surfaces via autonomous navigation of an inexpensive quadcopter. The description of the planes are not provided as input
• Enable unrolling of the scene to produce a panoramic mosaic of multiple planes

Fig. 1 provides an example. For the purpose of this paper, we assume that all the planes are visible from a single viewpoint of the quadcopter and planes are upright. To the best of our knowledge, this problem of autonomous mosaicing of upright multiplanar surfaces using quadcopter has not been considered before.

1.2 Challenges

The first challenge in solving this problem comes from the unknown geometry of the upright planes in the scene. Generally, a GPS is used for the navigation of unmanned aerial vehicles (UAVs). Cost consideration might have led our inexpensive quadcopter not to have a GPS, but that apart, we discount a GPS because it does not work effectively in indoor scenarios. Further, in the outdoor scenario, we do not make any assumption of the position of the scenes to be imaged. A GPS does not tell us the relative location of the craft with respect to the scene. In other words, the GPS is neither sufficient nor necessary for solving the problems we have considered.

Second, it is unreasonable to expect an inexpensive quadcopter to provide a crisp photograph from a specified location. Further, even a three-minute video captured by quadcopter, resulting in thousands of images overwhelm any mosaicing application. Hence, instead of videographing the whole area, the quadcopter should hover at certain points to take a stable video. Later, appropriate images from all videos are to be selected such that the whole scene is covered in those images. It becomes necessary to design an algorithm, which calculates those specific positions where the quadcopter should hover and record short videos.

Third, in the case of scenes with multiple planar surfaces, these problems worsen. Generally speaking, for concave surfaces (Fig. 2a), if the camera is placed at the centre of a concave surface, orthogonality can be achieved from a single viewpoint. However, if the plane is convex (Fig. 2b), viewpoints, as well as, the orientation have to be changed. The quadcopter has to recognise multiple planes in real-time and then change its direction for every plane before imaging the region so that it becomes normal to the plane.

Additional challenges arise from the requirement of inexpensive quadcopters. A quadcopter has an onboard inertial measurement unit (IMU) which consists of accelerometers, gyroscopes and magnetometers. The IMU provides pose which can be used to determine desired locations from where the whole scene can be covered. Due to the jerky nature of an inexpensive quadcopter, the IMU sometimes provides erroneous measurements, which results in incorrect pose information.

1.3 Contributions

Our goal is to image upright multiplanar regions autonomously using a quadcopter. Initially, three-dimensional (3D) positions of feature points on the imaging surface are estimated. These positions are used to detect multiple planes in real-time.

Next, we fly the quadcopter orthogonally to each plane, and within each plane, maneuver the craft on a small number of positions. The quadcopter hovers at each position for a small duration to capture the video of a part of the whole scene. Images from each plane are individually stitched together using a feature-based homography, i.e. we get individual mosaics per plane. Finally, we join individual mosaics using positional information from the quadcopter to create the unrolled mosaic. An example of the results is shown in Fig. 1. As mentioned earlier, we are not aware of any prior work for the problem specified with the constraints described.

2 Fundamental concepts

In this paper, we have provided an end-to-end application for autonomously imaging multiplanar regions. In doing so, we have leveraged on—with some adaptions—some fundamental techniques from computer vision, robotics and image processing; the basic concepts are mentioned here in the interests of readability.

2.1 Mosaicing

Image mosaicing is a process of combining two or more photographs of a scene into a single image, the idea being that the individual photographs are unable to do justice to the large scene with the existing field of view of either a camera or a human being. Individual photographs are expected to have some overlap; we first find features in each image, match these features between images, and then finally compute pairwise image warps to align them together. All methods assume the imaged scene is planar or that the camera has been rotated carefully around its centre of projection to avoid parallax. For more details, refer [1].

2.2 Path and motion planning

Path planning is an important step in the autonomous navigation of mobile robots. It is defined by sequence of ‘moves’ a robot should...
follow in order to reach desired destination from the source in an ‘optimal’ way. The optimal path may not necessarily mean the ‘shortest’ or the ‘fastest’ path as the optimality criteria change depending on the situation. Path-planning usually demands a map of the environment, and obstacles if any, and the robot needs to be aware of its location with respect to the map. If a map is available, one can use search techniques, or a brute force scan of the map to decide where to move next. For more details, refer [2].

While path planning defines the expected trajectory, the related term motion planning is related to change of position of the autonomous agent with respect to time. The actual command to the robot is needed; for instance, a robot could stop at a path to wait for another robot to cross a (different) path to avoid collisions. In this paper, the quadcopter hovers at calculated points to take pictures.

2.3 Simultaneous localization and mapping (SLAM)
As mentioned above, an autonomous agent requires both a map and a relative location. It turns out that these unknowns can be solved simultaneously with the SLAM technique. Prima-facie, SLAM is a hard problem as the two steps depend on each other resulting in an infinite loop. Many SLAM systems require a range measuring device such as LIDAR or RGB-D camera which can measure the depth of surrounding environment. If such devices are not available, one may use fusion of data from sensors such as accelerometers with visual input captured from RGB camera as discussed in this paper. For example, in visual SLAM one can start with a rough estimate of the 3D world, and repeatedly project the 3D world to 2D from the viewpoint of the robot to check if the image is consistent. In Section 4.1, a fusion approach involving sonar measurements, IMU and RGB images (without depth) is described for the problem at hand.

2.4 RANDom SAmple Consensus (RANSAC)
In the process of building a 3D map, 3D models such as planes need to be created from noisy features such as approximate 3D points. The RANSAC method is an iterative algorithm to estimate a mathematical model (such as a line, curve or plane) from error-prone data containing outliers. As the name of an algorithm suggests, the output of RANSAC can be different in different runs and the virtue of RANSAC is robustness.

3 Related work
Our work consists of two main quadcopter-based tasks: autonomous navigation and mosaicing. Hence, in this section, we discuss the state-of-the-art methods in each.

Autonomous navigation in robots is a well-studied subject. Two differences crop up in our problem. First, in our case, the images are captured aerially with an inexpensive quadcopter. The type of imagery, as well as the relative instability of the craft makes the SLAM problem harder. A technique named parallel tracking and mapping (PTAM) was introduced in [3] to estimate camera pose in an unknown scenario. This was substantially improved for the quadcopter case by Engel et al. [4] by fusing IMU data and the vision-based PTAM method. We rely on this method to obtain 3D feature points with the relative information of the 2D image, obviating the need for a GPS. This readily available method on the inexpensive quadcopter suggested itself in comparison to other more recent SLAM implementations such as the methods in [5–7].

We noticed a few problems though. While the scale accuracy of the craft is sufficient for simple navigation, the level of accuracy needed to position the craft for taking pictures was not attained. In the next section, we describe a simple modification that results in improved scale accuracy. With this modification, the method in [4] provides accurate positional information of the camera on the craft. Unfortunately, the roll and pitch information is not reliable due to jerky motion of our inexpensive quadcopter. Additionally, the 3D feature positions were (necessarily) approximate and sparse. Either the sparsity alone or the inaccuracy by itself, rule out seamless texture mapping in trying to create a high-quality mosaic. Likewise, the sparse 3D feature point also rules out any sort of methods from the structure from motion stable.

Panoramic image stitching (alternatively, image mosaicing) is a well-studied problem in the field of computer vision. Representative works include [8–13]. A full discussion is outside the scope of this paper, readers are referred to [14] for an excellent survey. Given the maturity of this area, there are various freeware, as well as, commercial software available for performing image stitching; most notable are AutoStitch [15], Microsoft’s Image Composer Editor [16], and Adobe’s Photoshop [17]. However, all these techniques work only when the scene lies on a single planar surface, or the scene lies on a cylindrical surface and camera is at the focus of that cylindrical surface. Also, these techniques provide reasonable results only when the camera is smoothly controlled by a human. As we would like to image large multiplanar surfaces using autonomous aerial vehicles, these conditions are not met.

Botterill et al. [18] and Kekec et al. [19] have developed methods for real-time mosaicing of aerial images captured by UAVs. Wischounig-Struc and Rinner [20] have used a system of multiple networked UAVs for incremental mosaicing of wide areas. Caballero et al. [21, 22] have used online mosaicing for improving the localisation of UAVs based on homographies computed from successive images. However, in these works, images are captured by the bottom camera of (expensive) UAV flying from very high altitudes (~100 m from the ground). These methods do not apply when high quality images of upright scenes captured with a frontal camera is desired.

Prasad et al. [23] have developed an imaging application with the use of an inexpensive quadcopter. This work builds on the earlier work. In [23], the navigation of the quadcopter is assumed to be manual. As explained above, manual navigation of quadcopter is considered inelegant and inappropriate. The glitch-prone performance of the manual control using the provided smartphone application can cause serious problems. The lag causes delays with the controls, thus risking a crash or a flyaway where the user has no idea what the craft is doing. Further, the algorithm in [23] assumes that the scene is made up of a single planar surface.

4 Methodology
The method adopted is pictorially depicted in the overview shown in Fig. 3 and is described in detail later on.

In brief, we probe the input scene (Fig. 3a) through a quadcopter, calculate 3D positions of feature points (Fig. 3b), and detect bounded multiplanar regions (Fig. 3c). Path planning is done for each bounded planar region to determine the camera positions (Fig. 3d). The quadcopter is autonomously manoeuvred along the computed path and videos are captured at target points. For each bounded planar region, the appropriate frame from each video is found (Fig. 3e) and then supplied to a mosaicing algorithm to create mini-panoramas for each plane. Finally, all mini-panoramas are joined using the pose information keeping the correct relative distance to get an unrolled view (Fig. 3f).

4.1 Scale accuracy
An essential step in autonomous navigation is for the quadcopter to ‘know’ what the geometry of the scene is. Because of the availability of excellent integration with Parrot AR.Drone, we used the software available [4] to (i) localise the quadcopter and (ii) to create a 3D map of the environment to be imaged (see Fig. 4). In this figure, we see potential feature points colour coded on the left. The 3D location of a subset of these is shown on the right from a different viewpoint. It may be noted that the points found are sparse and noisy. This poses a severe challenge in solving the problem posed.

The initialisation of the map is critical. The initialisation starts with a stereo-based algorithm, resulting in an unknown scale of the scene. Hence, there is a requirement of scaling the map to metric units. This is done by assuming that the camera is translated 10 cm between a stereo pair. This assumption may not be true in the case of an inexpensive quadcopter due to its jerky motions. Hence, proper scale estimation is necessary. In the sequel, we first briefly
describe the method in [4] in order to explain how we improved the scale accuracy.

The quadcopter measures the distance travelled during translation using both visual odometry, as well as the available metric sensors (ultrasound altimeter) at regular intervals. We get a pair of samples \( x_i, y_i \in \mathbb{R}^d \) where \( x_i \) is the presumed translation from the visual system and \( y_i \) is the distance measured by the metric sensor at each interval. These pair of samples are related according to \( x_i \approx \lambda y_i \). The scale \( \lambda \) is estimated by minimising the negative log-likelihood.

However, we cannot rely only on this method, since the visual feedback may lag due to, say, problems in wi-fi connectivity. Hence, there is a requirement of an alternative mechanism as a fallback. Engel et al. [4] have used the Extended Kalman Filter to fuse the metric observation model with the visual observation model for state prediction, in a unified way. That is, they estimate the pose \((x, y, z)\) and roll-pitch-yaw of the quadcopter at any given instant.

The state-space consists of a total of ten state variables.

\[
x_t(x, y, z, x, y, z, \Phi, \Theta, \Psi)^T \in \mathbb{R}^{10},
\]

where \((x, y, z)\) represents the position of the quadcopter in metric units and \((x, y, z)\) the velocity in m/s, both in world coordinates.
The state also contains three angles (in degrees), i.e. the roll $\Phi$, pitch $\Theta$, and yaw $\Psi$, of the quadcopter. An observation function $h(x)$ for each sensor as well as respective observation vector $z_t$ composed from the sensor readings are defined for each observation model.

A quadcopter measures its horizontal speed (i.e. along $x$ and $y$ direction) in its local coordinate system to get $x$ and $y$. The roll and pitch angles are directly taken from the accelerometers’ observations. Height measurements ($h_t$) and yaw measurements ($\Psi_t$) are differentiated and treated as observations of respective velocities. The resulting observation function $h(x)$ and measurement vector $z_t$ is given by

$$h(x) =
\begin{bmatrix}
x_t \cos \Psi_t - y_t \sin \Psi_t \\
x_t \sin \Psi_t + y_t \cos \Psi_t \\
z_t \\
\Phi_t \\
\Theta_t \\
\Psi_t
\end{bmatrix}
$$

(2)

$$z_t, f(E_{DC} E_C).$$

(5)

The pose estimate is scaled by the current estimate for the scaling factor $\lambda$ when the system tracks a video frame successfully. This pose estimate is transformed from the coordinate system of the front camera to the coordinate system of the quadcopter. Direct observation of the quadcopter’s pose is given by

$$h(x)(x_t, y_t, z_t, \Phi_t, \Theta_t, \Psi_t)^T$$

(4)

$$z_t, f(E_{DC} E_C).$$

(5)

where $E_{C_{ij}} \in \text{SE}(3)$ is the estimated camera pose (scaled with $\lambda$), $E_{DC} \in \text{SE}(3)$ the constant transformation from the camera to the quadcopter coordinate system and $f : \text{SE}(3) \rightarrow \mathbb{R}^6$ the transformation from an element of $\text{SE}(3)$ to our roll-pitch-yaw representation.

The extended Kalman filter is used to fuse all state variables from both observation models. Finally, the prediction model describes how the state vector $x_t$ evolves from one timestep to next. The quadcopter’s horizontal acceleration $\dot{x}$, $\dot{y}$ is approximated based on its current state $x_t$. The quadcopter’s vertical acceleration $\dot{z}$, yaw-rotational acceleration $\dot{\Psi}$, and roll and pitch rotational speeds, $\dot{\Phi}, \dot{\Theta}$, are estimated based on the state $x_t$ and active control command $u_t$. Finally, using quadcopter’s estimated pose information, we update the 3D positions.

4.1.1 Increasing scale accuracy: Although the above method provides a sparse 3D ‘map’ of the environment, the scale accuracy is less than what we desire. It leads to inaccurate 3D coordinates of feature points, which in turn results in wrong estimation of bounded planar regions. This in turn adversely affects the path planning resulting in incorrect camera positions.

To ensure accurate scale, we moved the quadcopter autonomously in the vertical direction (up and down) by a fixed distance. This up–down movement is repeated several times. The reason behind moving in the vertical direction is that the sonar sensor gives us accurate height information which is used to remove scale ambiguity. In contrast, the method described earlier [4] used two frames which are captured from positions 10 cm apart, but these are not in general positions, and not guaranteed to be vertically apart.

We conducted an experiment to demonstrate the efficacy of our method to achieve higher accuracy. The experimental setup is shown in Fig. 5.

As shown in Fig. 5, the quadcopter is placed at 2.3 m from the imaging plane for take-off. First, after take-off, we initialised the system using the method in [4], moved to the presumed location $(0, 0, k)$ (for different values of $k$) i.e. $k$ meters above the ground, and finally landed the quadcopter vertically down. Ideally, the take-off location and land location should be same. However, due to errors in the initialisation as well as noise in the IMU measurements, the distance between the two locations is non-zero. We use this distance as a measure of error.

Next, we introduced the up–down movement in the experiment, as per our algorithm. Our summarised observations after ten runs in both cases are listed in Table 1. Error in this table is the distance between the actual location of the quadcopter after landing, from the origin. Table 1 clearly indicates that the quadcopter can be considerably far off (around 85 cm) from the actual position if our algorithm is not employed.

A variation in the experiment is to dislocate the quadcopter (by a manual push), while it is hovering at the location $(0, 0, k)$. This was to check its ability to come back to the original position, a normal test done (and needed if the quadcopter has to work in outdoor conditions.) In the ‘dislocation’ variation of the experiment, there were cases when the quadcopter was not able to come anywhere close to $(0, 0, k)$.

The errors shown in either case may seem alarming. However, note that this is simply a manifestation of the use of a onetime experiment without the bundle adjustment for many frames and the extended Kalman filtering algorithm is done in obtaining depth.

4.2 Multiplanar regions versus multiple planes

Once we reliably possess metric information of feature points (roughly, this corresponds to Fig. 3b), it becomes necessary to figure out the boundaries of the region of interest (Fig. 3c). In particular, we have to detect multiple planes present in the user’s region of interest. Recall (see Fig. 4) that the method in [4] provides 3D points. Given a set of 3D points, the goal is to segment

![Fig. 5 Experimental setup to showcase the efficacy of our method to increase scale accuracy. The imaging plane is at a distance of 2.3 m from the take-off location of the quadcopter which attempts to hover vertically up](image)

Table 1 Scale correction

| Case         | Avg. estimated distance from plane ± std., m | Number of up–down movements | Mean error ± std., cm |
|--------------|---------------------------------------------|-----------------------------|-----------------------|
| earlier method | 1.2 ± 0.32                                 | NA                          | 82.6 ± 10.73          |
| our method   | 2.33 ± 0.01                                 | 3–4                         | 13.2 ± 3.6            |

The imaging plane used for this testing was 2.3 m away from the quadcopter (see Fig. 5). The second column indicates that the distance from the imaging plane estimated with our method (2.33 m) is close to the actual distance of the imaging plane. The last column reports where the quadcopter landed using the two methods when it was expected to land at the origin.
these points into planes and given planes, identify specific regions containing data to be imaged.

The problem of constructing multiple models such as multiple planes or multiple lines have been considered in the literature (e.g. sequential RANSAC [24], multiRANSAC [25], J-linkage [26], T-linkage [27]). Sequential RANSAC and multiRANSAC algorithms require the number of planes as an input parameter. However, as the number of planes changes according to the input scene, we cannot use these algorithms. In contrast, the method in [26] does not require the number of planes as an input parameter.

However, none of the methods provide the extent of the plane, i.e. the bounded region containing data. Fig. 6 depicts the relevant scenario. As we can see, there are several points that have ‘geometric’ affinity towards, say plane A. Algorithms based on RANSAC place these points in the cluster corresponding to plane A. However, in the real world these points are parts of plane B (with noisy depth) computed by the stereo algorithm. Hence, we have to disambiguate such points for the correct calculation of bounded multiplanar regions. We have developed an algorithm to modify the output in [26] to provide correct bounded multiplanar regions.

**Bounded planes**: Our algorithm to find continuous bounded multiplanar regions is given in Algorithm 1 (Fig. 7). This process is illustrated in Fig. 6. Fig. 8 shows a point cloud, as well as the multiplanar bounded regions estimated by our algorithm.

4.3 Path planning

At this point, we have bounded regions (Fig. 3c), and we need to create waypoints (Fig. 3d). The bounded region is divided into a grid of overlapping cells as shown in Fig. 9. Each cell is intended to correspond to a single image. We then use images from all cells to create a mosaic of a given bounded planar region. The cell dimensions (width, height) is decided on the basis of the amount of details required of the scene. For example, if we have to probe our scene with minute details, the quadcopter has to be much closer to the plane. Hence, in that case, the cell area is smaller. In all cases, we require an overlap between neighbouring images for successful mosaicing. The amount of overlap between two cells is a function of the necessary overlap in the feature space required for stitching.

Once we calculate coordinates of cell corners, we determine the desired position of a camera from where the whole cell area is covered in a single image. It is illustrated in Fig. 10. The coordinates of corner points are decided on the basis of cell width, cell height and the top-left corner of the cell. The image coordinates (in pixels) depends on the size of the image. We pose this problem as a point correspondence problem to estimate the position of the camera. We repeat this process for each cell to find out the waypoints of the path of a quadcopter to cover the region in optimal (in a number of positions) manner so that the whole region is captured.

Fig. 11 shows the computed path for each bounded region estimated in Fig. 8 with nodes indicating waypoints for the quadcopter to hover.

4.4 Navigation of a quadcopter

Once waypoints are determined, the complete navigation of the craft is determined as follows:

**Covering a bounded single planar region**: We navigate the quadcopter smoothly in a ‘snake scan’ manner along the target points.

**Transition from one plane to another plane**: Instead of moving directly ‘as the crow flies’ from the endpoint of one plane to the start of the new plane, we make sure that the transition in terms of yaw as well as the horizontal movement is smooth. To achieve this, we divide the horizontal distance into fixed parts and move along the x-axis and y-axis alternatively in a staircase manner. During each step, we change the yaw in equal proportion so that the...
quadcopter’s view is not changed drastically. Another reason behind this staircase-like movement is that the tracking method in [4] requires feature points to be detected in virtually every frame. During the transition between planes, there are typically far fewer feature points. The staircase movement ensures that the sparse feature points are present and the craft is not lost.

4.5 Recording video and ROS streams

We are now able to process the stage corresponding to Fig. 3d, and it is time to take photographs. Since the quadcopter is a quasi-stable flying craft, it is not prudent to take a photograph at each target position. Instead, we record a video on the USB device for a small amount of time (3 s) when the quadcopter is in the proximity of each marked point (Fig. 3d) in the calculated path. We also record ROS streams (image as well as positional data) and transmit over Wi-Fi to the host computer.

Finding sharp desired image: At this point (Fig. 3e), we have to find the desired sharp image, i.e. an image with minimum blur and which is closest to the target point from the 3-s video (~90 images) recorded on the USB device. There is no positional data available on the USB device. Positional data stream and image stream captured over Wi-Fi help us in this process. First, we synchronise these two streams captured using the timestamp information in the Wi-Fi stream. This gives us an approximate position of each image in the USB recorded video. Now, we select the sharpest image among the images captured over Wi-Fi which are taken from the positions in the proximity of the specified point. Later, we automatically match each image from the HD video (recorded on USB device) with the selected image (selected from the Wi-Fi image stream) using speeded-up robust features (SURF) [28] features. It gives us a subset of HD images which are within a threshold in SURF feature space from the selected image. Finally, we select the sharpest image from that subset of HD images.

4.6 Creating mosaics of the bounded multiplanar region

Once images for each bounded planar region are extracted from the captured videos (Fig. 3e), the mosaic for each planar region is created using the method of creation of a mini-panorama presented in [23]. Each mini-panorama represents one of the planar regions (e.g. one of the four images in Fig. 3f). However, we need to ‘attach’ these images in the manner shown with correct geographical gaps. Only when attached, do we get the complete ‘unrolled’ mosaic of the entire bounded multiplanar region as shown in Fig. 3f.

After the creation of the mini-panorama for each planar region, we join all mini-panoramas using a method described below. This is conceptually similar to the creation of super panorama used in [23] but there are important details. The super-panorama is created by forming a stereo pair between two images captured from the same depth, as shown in Fig. 12. Using the stereo disparity formula

\[ d = \frac{fb}{Z}, \]

where \( d \), \( f \), \( b \) and \( Z \) are disparity, focal length, baseline and depth, respectively, we can ‘place’, from the first viewpoint, the image captured from the second viewpoint, thereby creating a super-panorama. In [23], the disparity between the reference images of each mini-panorama is calculated by using the distance between the camera and the imaging plane, i.e. the depth and the distance between camera positions of reference images for each mini-panorama, i.e. the baseline.

In [23], as all mini-panoramas were in the same plane and a relatively simple stereo formulation (without rectification) was enough. In our case, as mini-panoramas belong to multiple planes, we have to project mini-panoramas on any one plane to find the disparity between reference images of all mini-panoramas. This step essentially involves image rectification. The path planning algorithm ensures that the quadcopter’s camera is normal to the
plane while capturing images from each planar region. Using this, the process of unrolling involves the calculation of the distance between the centre of projections and projection as shown in Fig. 13.

Let us assume that the depth of the camera from both the imaging planes is the same (say $Z$). Now, the disparity of the image on the second plane with respect to the image on the first plane is given as follows:

$$d = f \left( \frac{d_1 + d_2}{Z} \right), \quad (7)$$

where $d_1$ and $d_2$ are distances of projections of camera positions on the first and second plane, respectively from the line of intersection of two planes and $Z$ is the depth of the camera from the plane. If we compare (6) and (7), we see that now our baseline is $(d_1 + d_2)$, i.e. the distance between camera positions calculated along the plane, instead of calculating the direct distance between camera positions. The reason behind this difference is, imaging areas are on two different planes instead of one.

If the centre of projections are at different depths (e.g. say $Z_1$ and $Z_2$) first we bring the image captured from lesser depth (say $Z_1$) to the same depth of another image (which is captured from larger depth, say $Z_2$) by zooming out by the fraction $Z_1/Z_2$. Later, we use (7) to calculate the disparity by setting $Z = Z_2$.

To summarise our method (Fig. 3) first creates an accurate 3D map of the input scene spread over multiple planes using an extended initialisation step. Next, we determine the boundaries of each planar region. Further, we estimate the optimal number of positions along which we navigate the quadcopter. Later, we create the individual mosaics for each plane with the images extracted from videos. Finally, the individual mosaics are joined using super-panorama formulation to get an unrolled view of the input scene.

4.7 Optional: human interaction

In our experiments, we realised that it is possible that a user may be interested in only a particular part of the scene spread over multiple planes. In such cases, the user can mark the area of interest with the provided user interface.

Users are shown the live video stream as seen from the quadcopter (Fig. 14a). Now, users can click points to mark their area of interest on the 2D screen. At the backend, we determine the nearest 3D point corresponding to the location clicked by the user.
and then show its projection on the image plane (Fig. 14b). Once they finish marking the points, they can see the convex hull of the clicked points to check whether the area to be covered is indeed the area of interest (Fig. 14c). The user can optionally add or delete points or even completely remove all clicked points using the interface and select the new ones.

5 Experiments and results

All our experiments have been completed with the inexpensive consumer quadcopter called Parrot AR Drone 2.0. The camera resolution of AR Drone 2.0 is 1280 × 720 [When we stream images over Wi-Fi, the resolution drops to 640 × 360]. We have used the ROS-based ARDrone Autonomy Driver [29] to communicate with the quadcopter. We have also used tum ardrone package [30] for ROS to obtain the current state, i.e. (t, x, y, z, t, ˙x, ˙y, ˙z, Φ, Θ, Ψ, ˙Φ, ˙Θ, ˙Ψ)T (refer to equation (1)) of quadcopter. We have developed an ROS node for autonomous navigation of quadcopter. The code is developed in C++, and it is available as a GitHub repository, https://github.com/meghshyam/multiplanar. The mosaicking algorithm is also developed in C++ using the OpenCV library (OpenCV 2.4.9). Experiments were performed on a laptop with Intel Core i5 processor@2.4 GHz and 8 GB RAM.

We also took a picture of the scene, from a distance with a five mega-pixel camera to better understand the scene as well as to show the efficacy of our method.

5.1 Single plane with multiple visits

We have first used our algorithm to image a single planar wall shown in Fig. 15a. As the wall was too tall and it was not possible to view the full wall in one visit, we selected the desired area in two steps as shown in Figs. 15b and c. For path planning stage, the overlap is set at 30% of the cell area to deal with sparser (containing very few feature points) images. In path planning, 37 positions (12 for the top part and 25 for the bottom part) were estimated to encompass the user-selected area. The quadcopter is autonomously manoeuvred along these positions. All frames from the videos captured while navigating along these locations are shown in Fig. 16a. Our algorithm picks images (from all frames) captured from 37 positions (shown in Fig. 16b). These images are further mosaiced to get the final output as shown in Fig. 15c.

5.2 Multiple planes

Further experiments are done on various setups covering multiple planes. 

Concave: In this experiment, the exhibits were arranged in the concave fashion as shown in Fig. 17a. We have selected the area to be imaged as shown in Fig. 17b. In the path planning stage, overall 27 (9 from the left plane, 12 from the middle plane, and 6 from the right plane) positions were estimated to encompass the user-selected area. Images captured from those positions are mosaiced using our algorithm to get the final output as shown in Fig. 17c.

The accuracy of our panorama can be verified from Table 2 where the area of each bounded region (in metric dimensions), as well as, that of corresponding mini-panorama (in pixels) are given. In Table 2, it can be seen that physical area of each bounded region (shown in columns 2, 4 and 6) is in proportion with the size of the corresponding mini-panorama (shown in columns 3, 5 and 7, respectively).

As a numerical verification, the aspect ratio is provided in the last (third) row.

Convex: In this experiment, paintings were arranged in the convex fashion as shown in Fig. 18a. We have selected the area to be imaged as shown in Fig. 18b. In the path planning stage, overall 30 (9 from the left plane, 12 from the middle plane, and 9 from the right plane) positions to encompass user-selected area are estimated. Images captured from those positions are mosaiced using our algorithm to get the final mosaicing output as shown in Fig. 18c.

Mixed: We have performed an experiment where the posters were arranged in the mixed fashion. The arrangement looked like Fig. 19a where the middle posters form the convex region while the side posters form the concave region. The selected area for imaging is shown in Fig. 19b. In path planning overall 24 (six from each plane) positions are estimated to encompass the user-selected area.
Images captured from those positions are mosaiced using our algorithm to get the final output as shown in Fig. 19.

Planes at different depth: In this experiment, we have arranged posters parallel to each other, but at different depths, as shown in Fig. 20a. It is not possible to mosaic them together due to change in planes. However, we have imaged each exhibit independently using quadcopter and brought the mosaic of the exhibit at a larger depth to the same depth of lesser depth exhibit mosaic using the estimated plane equations’ information. The final result is shown in Fig. 20c.

---

**Fig. 16** Imaging of a large planar wall and selection of images
(a) Frames from videos captured by the quadcopter at computed locations,
(b) 37 images selected by our algorithm as necessary for the mosaicing

**Fig. 17** Art exhibition
(a) Three different artwork arranged in a concave fashion and shown in the long-range photograph appear with differing details; we wish to see a consistent view,
(b) Green quadrilateral shows the area selected by the user, while the blue, violet and magenta coloured quadrilaterals represent the corresponding multiple bounded planar regions estimated by our algorithm,
(c) Images captured by our autonomous navigation algorithm are mosaiced

**Table 2** Comparison of the physical dimensions of the multiplanar bounded regions and the corresponding generated mini-panoramas

| Plane  | Mini-panorama  | Plane  | Mini-panorama  | Plane  | Mini-panorama  |
|--------|----------------|--------|----------------|--------|----------------|
| 1      | 1.16 m × 0.7m  | 2100 × 1300 | 1.18 m × 0.7m  | 2150 × 1300 | 1.13 m × 0.7m  | 2050 × 1300 |
| aspect ratio | 1.66          | 1.62      | 1.69            | 1.65      | 1.61            | 1.58         |

The aspect ratios of the respective mini-panoramas are, as desired, matching.
6 Concluding remarks

We have developed an end-to-end application for autonomously imaging multiplanar regions using quadcopter. We have also developed an algorithm for ‘unrolling’ the multiplanar scene using the fusion of IMU data and video captured by a quadcopter.

In our solution, we autonomously manoeuvre a quadcopter along the planned path to capture each plane in the multiplanar surface. The path planning for each plane is done to estimate optimal locations such that images from the estimated locations will cover the whole plane. Later, we stitch images for each plane to create a mini-panorama. Finally, mini-panoramas are merged using positional information to form a full panorama. Our method works in various planar setups such as disconnected, convex, concave as well as mixed.

One limitation of our approach is that it works only when the multiplanar surfaces are all simultaneously visible from some viewpoint. In the future, the method can be extended for the

7 References

[1] Szeliski, R.: ‘Computer vision: algorithms and applications’ (Springer, London, UK, 2011)
[2] LaValle, S.M.: ‘Planning algorithms’ (Cambridge University Press, Cambridge, England, 2006)

[3] Klein, G., Murray, D.: ‘Parallel tracking and mapping for small AR workspaces’. Proc. of 6th IEEE and ACM Int. Symp. on Mixed and Augmented Reality, Nara, Japan, 2007, pp. 1–10

[4] Engel, J., Sturm, J., Cremers, D.: ‘Camera-based navigation of a low-cost quadrocopter’. Int. Conf. on Intelligent Robots and Systems, Algarve, Portugal, 2012, pp. 2815–2821

[5] Yang, S., Song, Y., Kaess, M., et al.: ‘Pop-up SLAM: semantic monocular plane SLAM for low-texture environments’. Int. Conf. on Intelligent Robots and Systems, Daegu, Korea, 2016, pp. 1222–1229

[6] Engel, J., Schöps, T., Cremers, D.: ‘LSD-SLAM: large-scale direct monocular SLAM’. European Conf. on Computer Vision, Zurich, Switzerland, 2014, pp. 834–849

[7] Mur-Artal, R., Montiel, J.M.M., Tardós, J.D.: ‘ORI-SLAM: a versatile and accurate monocular SLAM system’, IEEE Trans. Robot., 2015, 31, (5), pp. 1147–1163

[8] Miligram, D.: ‘Computer methods for creating photomosaics’, IEEE Trans. Comput., 1975, C-24, (11), pp. 1113–1119

[9] Miligram, D.: ‘Adaptive techniques for photomosaicning’, IEEE Trans. Comput., 1977, 26, (11), pp. 1175–1180

[10] Capel, D., Zisserman, A.: ‘Automated mosaicing with super-resolution zoom’. IEEE Conf. on Computer Vision and Pattern Recognition, Santa Barbara, USA, 1998, pp. 885–891

[11] Szeliski, R., Shum, H.-Y.: ‘Creating full view panoramic image mosaics and environment maps’. ACM SIGGRAPH, Los Angeles, USA, 1997, pp. 251–258

[12] Brown, M., Lowe, D.G.: ‘Automatic panoramic image stitching using invariant features’, Int. J. Comput. Vis., 2007, 74, (1), pp. 59–73

[13] Brown, M., Lowe, D.: ‘Recognising panoramas’. Int. Conf. on Computer Vision, Nice, France, 2003, pp. 1218–1225

[14] Szeliski, R.: ‘Image alignment and stitching: A tutorial’, Tech. rep., MSR-TR-2004-92, Microsoft Research, 2005

[15] Autostitch: ‘Last visited May 5 2019’ Available at http://www.cs.bath.ac.uk/brown/autostitch/autostitch.html

[16] Microsoft Image Composite Editor: ‘Last visited May 5 2019’ Available at http://research.microsoft.com/en-us/um/redmond/groups/cvms/ICE/

[17] Adobe Photoshop CS6: ‘Last visited May 5 2019’ Available at http://www.adobe.com/products/photoshop

[18] Botevirr, T., Mills, S., Green, R.: ‘Real-time aerial image mosaicing’. Int. Conf. Image and Video Computing, Queenstown, New Zealand, 2010, pp. 1–8

[19] Kekeci, T., Yildirim, A., Unel, M.: ‘A new approach to real-time mosaicing of aerial images’, Robot. Auton. Syst., 2014, 62, pp. 1755–1767

[20] Wischounig-Strucel, D., Rinner, B.: ‘Resource aware and incremental mosaics of wide areas from small-scale UAVs’, Mach. Vis. Appl., 2015, 26, (7), pp. 885–904

[21] Caballero, F., Merino, L., Ferruz, J., et al.: ‘Improving vision-based planar motion estimation for unmanned aerial vehicles through online mosaicing’. IEEE Int. Conf. on Robotics and Automation, Orlando, USA, 2006, pp. 2860–2865

[22] Caballero, F., Merino, L., Ferruz, J., et al.: ‘Unmanned aerial vehicle localization based on monocular vision and online mosaicing’, J. Intell. Robot. Syst., 2009, 55, (4), pp. 323–343

[23] Prasad, M.G., Chandran, S., Brown, M.S.: ‘Mosaicing scenes through quadcopter’. IEEE Winter Conf. on Applications of Computer Vision, Lake Placid, USA, 2016, pp. 1–8

[24] Kanazawa, Y., Kawakami, H.: ‘Detection of planar regions with uncalibrated stereo using distributions of feature points’. British Machine Vision Conf., London, UK, 2004, pp. 1–10

[25] Zuliani, M., Kenney, C.S., Manjunath, B.S.: ‘The multiRANSAC algorithm and its application to detect planar homographies’. Int. Conf. on Image Processing, Genoa, Italy, 2005, vol. 3, pp. 153–156

[26] Toldo, R., Fusiello, A.: ‘Robust multiple structures estimation with J-linkage’. European Conf. on Computer Vision, Marseille, France, 2008, pp. 537–547

[27] Magri, L., Fusiello, A.: ‘T-Linkage: a continuous relaxation of J-linkage for multi-model fitting’. IEEE Conf. on Computer Vision and Pattern Recognition, Columbus, Ohio, 2014, pp. 3954–3961

[28] Bay, H., Tuytelaars, T., Gool, L.V.: ‘SURF: speeded up robust features’. European Conf. on Computer Vision, Graz, Austria, 2008, pp. 404–417

[29] ROS Driver for ARDrone Autonomy: ‘Last visited May 5 2019’ Available at https://github.com/AutonomyLab/ardrone_autonomy

[30] ROS package for tum_ardrone: ‘Last visited May 5 2019’ Available at http://wiki.ros.org/tum_ardrone