AERIAL EYES - QUADCOPTER IMPLEMENTING PANORAMA AND 3D RECONSTUCTION

by

AVINASH RAMESH 2012103009
MITHUL MATHIVANAN 2012103037
SHRIYA SASANK 2012103069

A project report submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANNA UNIVERSITY, CHENNAI – 25

MAY 2016

BONAFIDE CERTIFICATE

Certified that this project report titled **AERIAL EYES - QUAD-COPTER IMPLEMENTING PANORAMA AND 3D RE-CONSTUCTION** is the *bonafide* work of **AVINASH RAMESH (2012103009)**, **MITHUL MATHIVANAN (2012103037)** and **SHRIYA SASANK (2012103069)** who carried out the project work under my supervision, for the fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place: Chennai
Date:

Dr. AP Shanthi
Professor
Department of Computer Science and Engineering
Anna University, Chennai – 25

COUNTERSIGNED

Head of the Department,
Department of Computer Science and Engineering,
Anna University Chennai,
Chennai – 600025

ACKNOWLEDGEMENT

We would like to thank the entire committee consisting of Dr. V. Vetriselvi, Dr. S. Renugadevi, Dr. P. Velvizhy for constantly motivating us and identifying the areas of improvement on the project. We would like to extend our immense gratitude to our project guide, Dr. AP Shanthi for her perpetual support and able guidance which was instrumental in taking the project to its successful conclusion. Finally, we would like to thank the Head of the department, Dr. D. Manjula for providing a conducive environment and amenities to facilitate our project work.

Avinash Ramesh **Mithul Mathivanan** **Shriya Sasank**

ABSTRACT

Quadcopters are one of the most popular technologies of the 21st century. They have multifarious uses ranging from aerial photogrammetry(photography and 3d positioning of objects), military surveillance, taking panorama shots, movie-making and delivery of products. It is also an extensive area of research, an example of which would be swarm robotics. Our project delves into aerial photogrammetry i.e. taking pictures of objects/topography and reconstructing 3D representations of it. Since it is extremely difficult to take pictures or videos of places inaccessible to humans, such as caverns, waterfalls, ancient ruins, caves, etc. Such places can be observed and examined with the help of drones. However very few drones have the stability required to provide stable video for processing or images without excessive noise to generate a panorama image - a representation of a 360 degree angle of physical space. The drone would be able to take pictures of objects in inaccessible places and reconstruct 3D models of it for obeservation, analysis and study.

ABSTRACT

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

RPi	Raspberry Pi
IDE	Interactive Desktop Environment
UAV	Unmanned Aerial Vehicle
PWM	Pulse Width Modulation
IMU	Inertial Measurement Unit
PID	Proportional-Integral-Derivative
GUI	Graphical User Interface
GPS	Global Positioning System
JSON	JavaScript Object Notation
WiFi	Wireless Fidelity
SfM	Structure from Motion
DOF	Degrees Of Freedom
BLDC	BrushLess Direct Current
ESC	Electronic Speed Controller
SD	Secure Digital
LiPo	Lithium Polonium
RC	Radio Control
UI	User Interface

CHAPTER 1

INTRODUCTION

1.1 GENERAL OVERVIEW

The Quadcopter UAV is one of the fastest growing technologies of the 21st century. It has a myriad of uses and is a field of intensive research and continuous development. The broad objective of this project is to build a drone which would be able to take pictures of topography or objects, construct a panorama (360 degree view of the space) and recreate a 3D model of the environment. It is a highly useful technology as it can be used in aerial photogrammetry, reconstructing topographical landscapes for archaeological excavations, reconstruction of buildings and taking panorama shots for movie-making etc. The quadcopter is built from scratch and runs on the Raspberry Pi microcomputer with a Python interface. The image processing is done on-board the craft using OpenCV 3.1.0. The flight dynamics of the quadcopter are automatically stabilized using a PID controller coded in Python. The drone is controlled manually using an Android App with a virtual joystick interface for easy control. The drone's flight and functionality is controlled via the App and the processing of images to produce the panorama and 3D model is done onboard by the Raspberry Pi. The visualization of the 3D reconstuction is viewed on the laptop.

1.2 ABOUT THE PROJECT

The project is divided into three main modules. The first module is the quadcopter itself. The quadcopter runs on the Raspberry Pi 2 microcomputer which hosts a Debian-based OS (Raspbian Jessie) which allows the use of OpenCV and Python. It has four motors whose speeds are controlled using Pulse Width Modulation(PWM) technique. PWM signals are used to communicate with the ESC which controls the voltage given to the BLDC motors and thus control the speed. The RPi uses pigpio library to issue PWM signals to the ESC. The flight dynamics including stability and orientation are detected using the IMU(Inertial Measurement Unit) Sensor which consists of an accelerometer, gyroscope and magnetometer. The IMU sensor has 9 degrees of freedom since the accelerometer, gyroscope and magnetometer have 3 axes each. The accelerometer measures the acceleration and tilt while the gyroscope measures the angular velocity. The sensor data are fed to the RPi controller which calculates the current orientation and automatically computes the required orientation of the quadcopter using a PID algorithm. The RPi runs a Python server which receives JSON-encoded altitude, motor speed and trim values which are decoded and used as input by the Quadcopter program. The quadcopter is further equipped with a camera module which takes pictures and feeds them to the RPi's image-processing program which then performs Panorama stitching and 3D Reconstruction of the captured images. The second module performs image processing on the images captured by the RPi camera. It is divided into two parts Panorama Stitching and 3D Model Reconstruction. The module has been implemented using OpenCV 3.1.0 with Python support. The quadcopter takes pictures using the camera module which are fed to the RPi. OpenCV is used to stitch the images together to create a panorama or a 360

degree view of the environment. Panorama stitching allows one to take individual images of a wide physical space which cannot be covered in a single shot, finds overlapping features, matches them and stitches them into one large, combined shot. This can be used in various applications such as movie-making, aerial photogrammetry and terrain localization and mapping. The second part of the module deals with 3D reconstruction. The reconstruction algorithm also called Structure from Motion(SfM) algorithm is used to recreate 3D point clouds of buildings or topographies whose pictures are taken by the quadcopter. The point clouds are then visualized on a laptop using a browser. The advantage of using a quadcopter for this purpose is that it can fly to inaccessible locations and take shots which would otherwise be very difficult to capture. It can therefore be used for archaeological surveys, security and dynamic reconstruction of terrain. The third module is the Android App which is used to control the quadcopter. Usually quadcopters are controlled using Radio Controllers but this project makes use of an Android app which will have other functions apart from simply controlling the flight of the drone viz. sending JSON-encoded signals using WiFi to the Python server running on the RPi. The app allows the user to manually control the altitude of the quadcopter, change trim values(which are special values used for stabilizing the quadcopter) and additionally control the speed of individual motors. The app has a virtual joystick interface for simple user-interaction and control.

CHAPTER 2

RELATED WORK

A significant amount of research and work has gone into the quadcopter. There have been several algorithms for controlling the stability of the quadcopter such as PID(Proportional-Integral-Derivative) and Fuzzy Logic. With improvements to open source image processing libraries like OpenCV and the development of RaspberryPi microcomputer, the capability and use of drones for various activities have increased. Alex G. Kendall, Nishaad N. Salvapantula and Karl A. Stol developed an on-board object-tracking control of a quadcopter with monocular vision[1]. This project used a Raspberry Pi for processing along with a camera module and sensors as input to the OpenCV library for object tracking based on the color of the objects. While our project also makes use of the RPi and OpenCV for image processing, it does not perform object tracking. It takes pictures which are then used for panorama stitching and 3D model reconstruction. Their quadcopter could detect the relative positions of static objects but could not dynamically follow objects. Our project does not deal with tracking and we have focused largely on the flight stability and image processing modules.

Quadcopter control has mostly been through Radio Control joysticks. Since Android phones are ubiquitous now, we developed a virtual joystick interface to control the flight of the quadcopter as an Android app. The app is also used for directing image-processing functions of the drone. Since our quadcopter's functions are panorama stitching and

3D reconstruction, we needed to develop an interface for rendering the results. A. Zul Azfar and Hazry Desa[2] had developed a Graphical User Interface for monitoring the flight dynamics and orientation of a remotely-operated quadcopter. The GUI was developed using LABView software. Basically, the GUI on the PC accurately captured the motion of the quadcopter as images which were rendered using LabVIEW software. The quadcopters processing was done on-board using Arduino microcontroller and the balancing and stability were managed using an IMU sensor. This was a path-breaking paper as it demonstrated how an unmanned aerial vehicle could be controlled from a remote station. The project did not concern itself with the stability of the drone or perform any specialized functions. Its orientation could be rendered on a GUI. Priyanga M. And Raja Ramanan [3] developed a UAV for Video Surveillance to prevent unauthorized soil-mining operations. It used a RPi for on-board processing of live feeds from a webcam. The videos could be viewed on a webpage in real-time using WiFi. The drone used a GPS to calculate position and follow the target. The project demonstrated how a drone could be used for practical and real-time applications such as surveillance and tracking with high accuracy. In our project, signals from the app are sent as JSON-encoded data using WiFi to the Python server running on the quadcopter and corresponding results from image processing/pictures aboard the drone are rendered in a browser interface by a laptop on the ground. Since our project does not involve object-tracking, it does not use a GPS receiver but the quadcopter itself is controlled remotely via WiFi using the Android App. Image processing on-board the quadcopter has been implemented by K.S. Shilpashree, Lokesha H and Hadimani Shivkumar [6] in their paper, Implementation of Image Processing on Raspberry Pi. They have described the purpose of applying image-enhancing algorithms to improve the quality of the

images taken by the drone since the quality of the images may be compromised due to unstable flight. Image-enhancing algorithms such as Rudin-Osher-Fatemi de-noising model (ROF) have been implemented. We have implemented Panorama stitching and 3D reconstruction aboard the drone. Due to the heavy processing required, the RPi's speed in performing 3D Reconstruction is a little slow.

Shawn McCann's thesis on 3D Reconstruction from Multiple Images [5] has tried to identify various techniques for dense and sparse reconstructions using Structure from Motion(SfM) algorithms.He has demonstrated and implemented the complete flow of 3D Reconstruction using the open source Bundler software. Our project makes use of VLFeat which has SfM support and OpenCV which offers a variety of useful libraries for feature detection and matching. In both projects, Bundle Adjustment module was used to minimize reprojection errors and generate sparse point clouds. While [5] rendered the point clouds using third-party tools(MeshLab), our project uses an interactive JS library called, threeJS which renders the point cloud on a browser.

Drone technology has advanced enormously along the lines of photography, surveillance, exploration and mapping of terrains and delivery. We were inspired by the fact the mobility and usability of drones juxtaposed with the computational power, portability and open-source support of the RPi from these projects and hence, decided to build one which combined practical and useful image-processing applications.

CHAPTER 3

REQUIREMENTS ANALYSIS

3.1 REQUIREMENTS

3.1.1 Quadcopter

Hardware Requirements for the Quadcopter

1) Raspberry Pi 2 microcomputer
2) 9DOF IMU Sensor
3) 4 BLDC Motors
4) Raspberry Pi Camera Module
5) 4 Carbon Fibre Propellers
6) Battery
7) Quadcopter Frame
8) ESC
9) Male-Female Connecting Wires
10) 4 Propeller Guards
11) Breadboard
12) Resistors
13) Portable Charger

Software Requirements for the Quadcopter

1) Raspbian OS Jessie
2) OpenCV 3.1.0 and OpenCV 3.1.0
3) Python
4) Android Studio

Functional Requirements for the Quadcopter

1) Quadcopter should be connected to the WiFi network
2) LiPo batteries should be charged
3) The SD Card should be securely connected
4) The SD Card should have enough space
5) Quadcopter should move according to the processed flight control signals

3.1.2 Image Processing

Hardware Requirements for the Image processing module

1) Raspberry Pi Camera
2) Raspberry Pi microcomputer (Processing unit)

Software Requirements for the Image processing module

1) Python 2.7.6
2) OpenCV 3.1.0

Functional Requirements for Image processing module

1) The Raspberry Pi should be connected to the WiFi network
2) The Raspberry Pi Camera should be mounted securely on the quadcopter
3) The quadcopter must be stable enough to take fairly clear pictures
4) The camera signal from the Android app should be received by the quadcopter
5) The SD Card must have enough space to store the images taken
6) The Raspberry Pi must execute the Python program without system failure

3.1.3 Android App Module

Software Requirements for the Android App module

1) Android Studio IDE
2) Terminal emulator for android

Hardware Requirements for the Android App module

3) Android phone supporting minimum API 18

Functional Requirements for the Android App module

1) Create the Mobile Hotspot
2) Send motion, orientation and camera signals from the app to the Raspberry Pi

3.1.4 Analysis

The quadcopter is built using the Raspberry Pi. The Raspberry Pi was used beacuse of its computational power, portability and immense opern-source support. The quadcopter is built with 4 motors, propellers, an IMU sensor, ESC and a Raspberry Pi native camera that is interfaced with the Raspberry Pi. The Android app sends JSON-encoded signals to the Pi.The Raspberry Pi hosts a Python server which decodes the signals received and triggers the appropriate Python program to fly the quadcopter and take pictures. The IMU sensor detects the orientation of the quadcopter and feeds that data to the quadcopter balancing program. The program can then use the roll,pitch and yaw data to compute the required orientation of the quadcopter. The Pi Camera takes images which are processed using OpenCV image processing library on the Pi and saved on the SD Card.

CHAPTER 4

SYSTEM DESIGN

4.1 COMPONENTS IN THE PROJECT

4.1.1 Hardware Components

1. Raspberry Pi 2 microcomputer
2. 9DOF IMU Sensor
3. 4 BLDC Motors
4. Raspberry Pi Camera Module
5. 4 Carbon Fibre Propellers
6. Battery
7. Quadcopter Frame
8. ESC
9. Male-Female Connecting Wires
10.4 Propeller Guards
11.Breadboard
12.Resistors
13.Portable Charger

4.1.2 Software Components

1. Raspbian OS Jessie
2. OpenCV 3.1.0
3. Python 2.7.6
4. Android Studio IDE

4.1.3 Programs

a) Python programs to control the quadcopter:

1. PID Controller
2. Kalman Filter
3. Quadcopter Control
4. Python Server

b) Python programs for Panorama Stitching:

1. Camera Calibration
2. Panorama Stitching

c) Python Programs for 3D Reconstruction:

1. Extract Metadata
2. Detect Features
3. Match Features
4. Create Tracks
5. Reconstruct
6. Mesh

d) HTML Code for Visualizing Sparse Reconstruction

e) Android App providing virtual joystick interface

4.2 TECHNOLOGY USED

4.2.1 Raspberry Pi 2 Microcomputer

The quadcopter is controlled using the Raspberry Pi 2. Raspberry pi (RPI) is a credit card sized microcomputer that runs on a Linux based operating system loaded on its SD card . It includes a 900MHz quad-core ARM Cortex-A7 CPU and 1GB RAM. It can connect to a network by using an external user-supplied USB Ethernet or WiFi adapter. Furthermore, the RPi has four USB ports, a full HDMI port, 40 GPIO pins and an Ethernet port. Apart from this, the Raspberry

Pi has a camera interface which allows a native Pi Camera to be configured and used. It also has a Micro SD Card slot. The advantage of using a RPi over traditional Arduino boards is that the Pi has much greater computational power and thus, more complicated and better stabilization algorithms can be run on it.

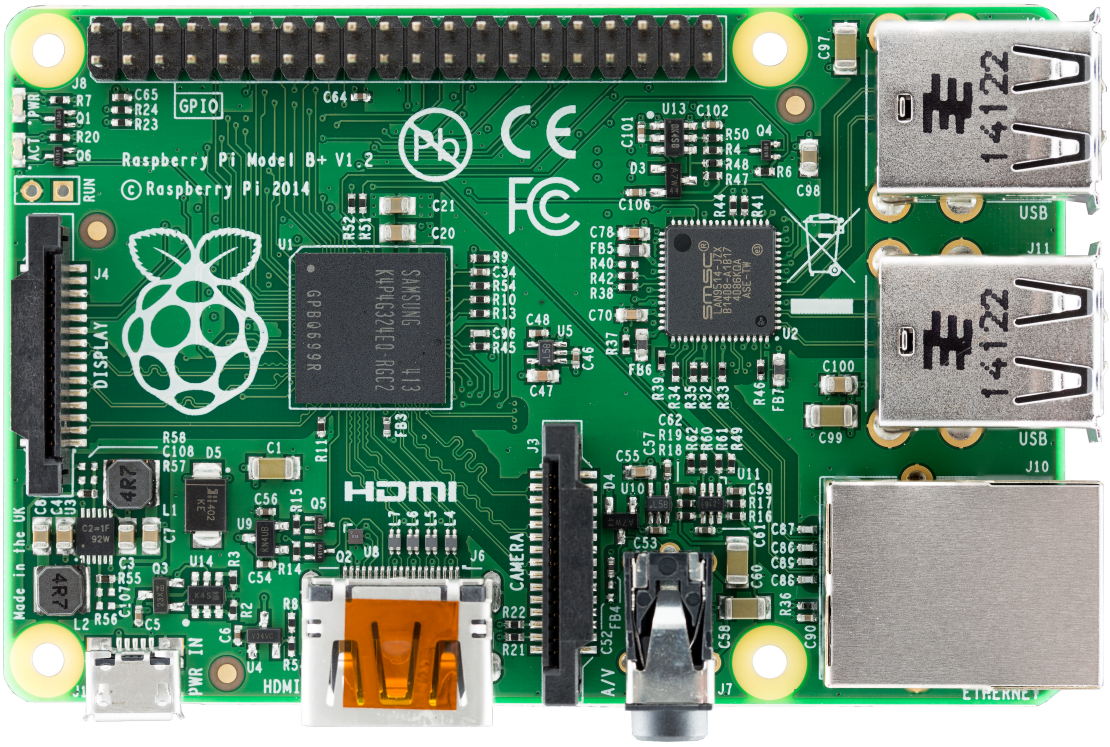

Figure 4.1: Raspberry Pi

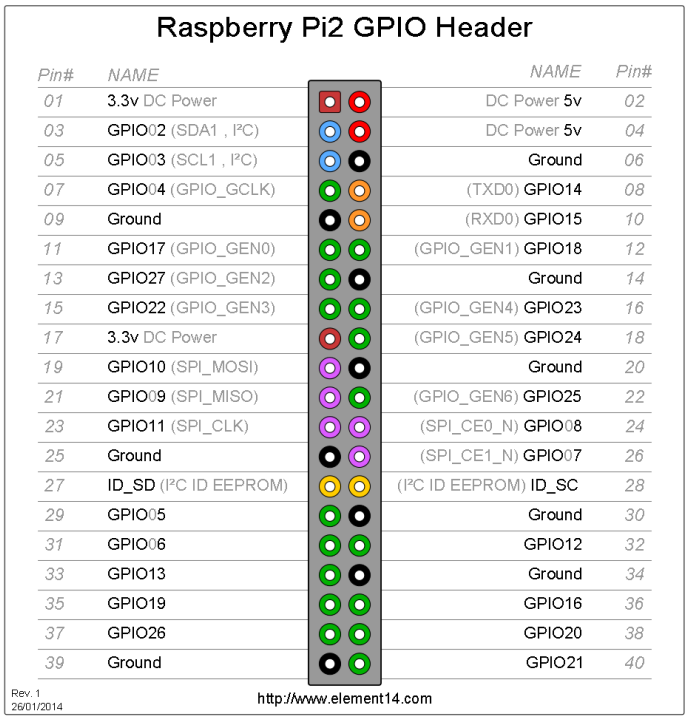

Figure 4.2: Raspberry Pi 2 GPIO Header

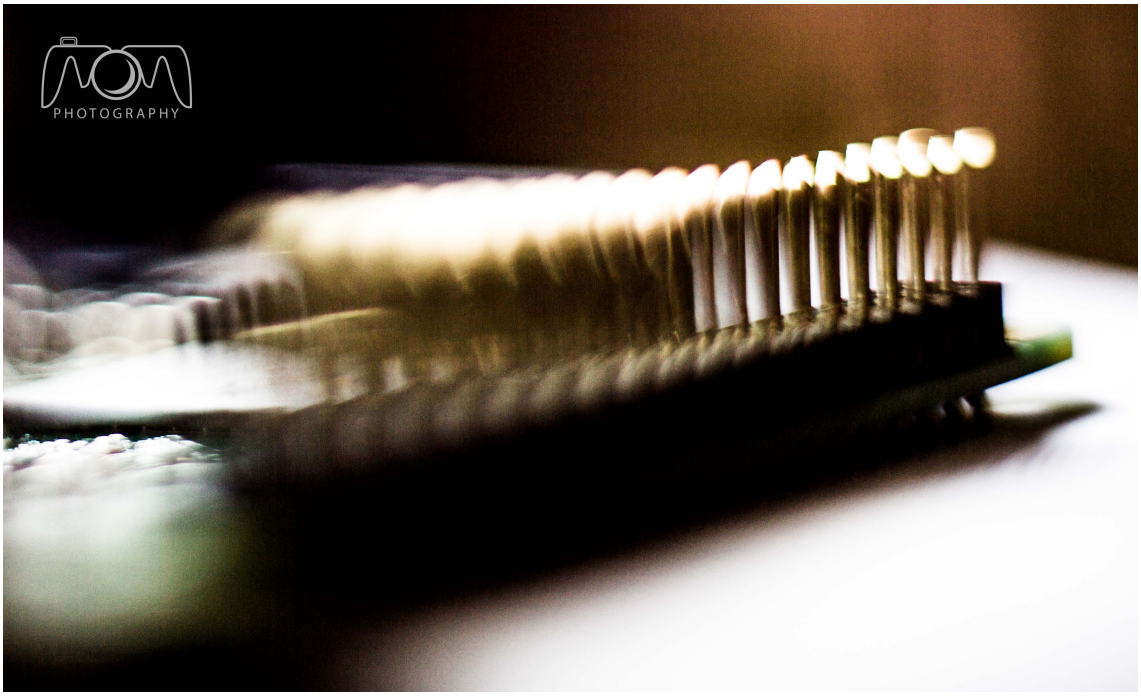

Figure 4.3: Raspberry Pi 2 GPIO Pins

4.2.2 Electronic Speed Controller (ESC)

The Electronic Speed Controller is a circuit which varies the motor's speed, direction and acts like a dynamic brake. ESCs are most often used with BLDC motors necessary for translating digital signals to electric current. ESCs are an essential component of modern multirotor crafts that offer high power, high frequency, high resolution 3-phase AC power to the motors in an extremely compact miniature package. These crafts depend entirely on the variable speed of the motors driving the propellers. The large variation and fine RPM control in motor/propeller speed gives all of the control necessary for a quadcopter (and all multirotors) to fly. On a quadcopter, each motor gets its own ESC, each of which connects to the controller. After computing the inputs(the amount of current and direction to be given to each motor), the controller directs each ESC to adjust its speed according to the PWM signal provided. This quadcopter uses a 40A, 3s ESC.

Figure 4.4: 40A 3s ESC

4.2.3 IMU Sensor

An inertial measurement unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the magnetic field surrounding the body, using a combination of accelerometers and gyroscopes and magnetometers. IMUs are typically used to maneuver aircraft, including UAVs. This project uses a 9DOF IMU sensor with a 3-axis accelerometer, 3-axis gyroscope and 3-axis magnetometer. The accelerometer measures acceleration and has high accuracy. The gyroscope measures the angular velocity and is used to stabilize flight. It is less accurate than the accelerometer and is prone to drift errors.

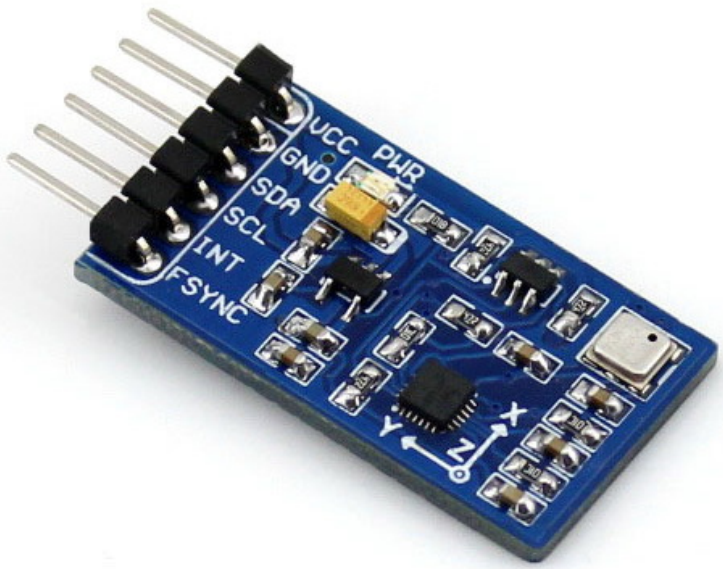

Figure 4.5: IMU Sensor

4.2.4 Raspberry Pi Camera

The Raspberry Pi camera module can be used to take high-definition video, as well as photographs. The camera has a five megapixel fixed-focus camera that supports 1080p30, 720p60 and VGA90 video modes, as well as stills capture. It attaches via a 15cm ribbon cable to the CSI port on the Raspberry Pi. It can be configured using the Python Picamera library. The project uses the Pi Camera to capture images for panorama stitching and 3D reconstruction.

Figure 4.6: Raspberry Pi camera

4.3 BLOCK DIAGRAM

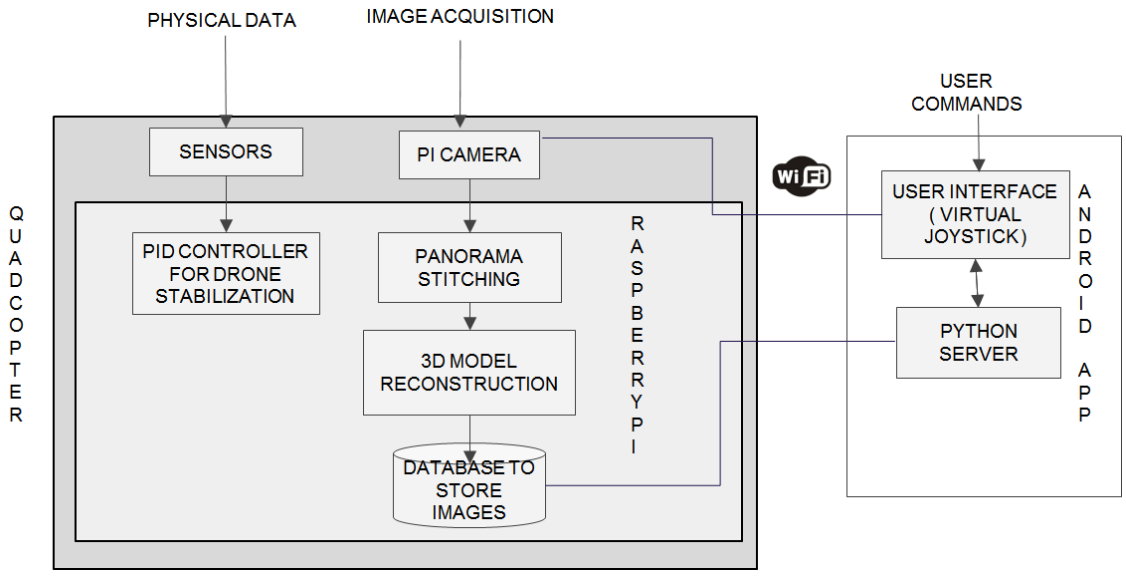

Figure 4.7: Block Digram

4.4 ARCHITECTURE DIAGRAM

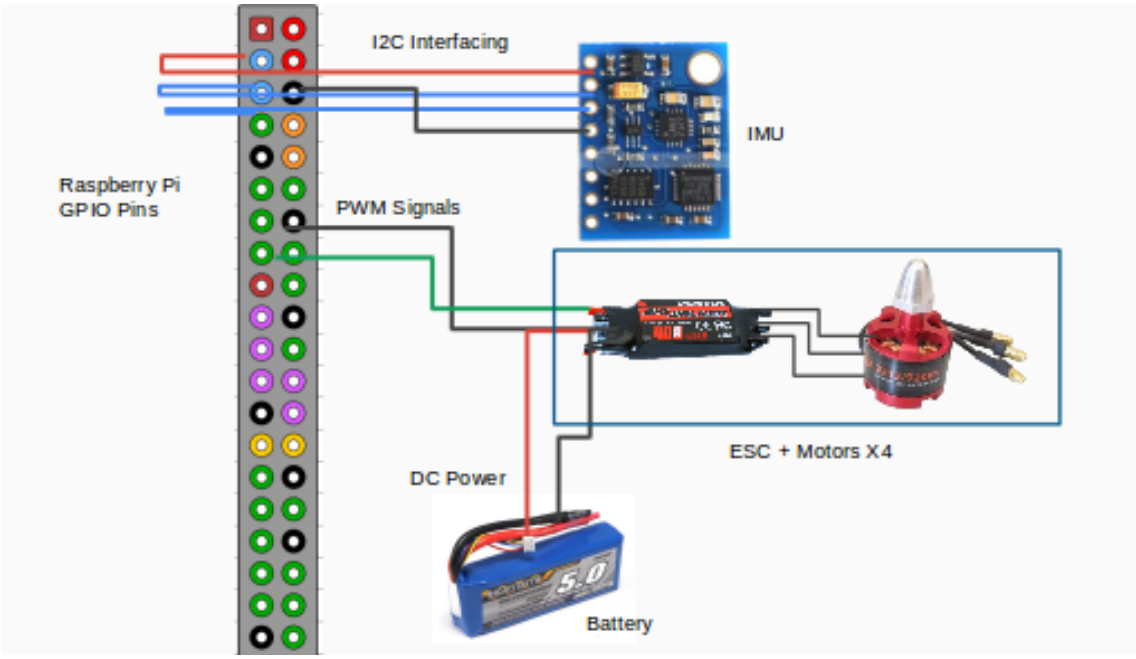

Figure 4.8: Circuit Digram

4.5 CIRCUIT DIAGRAM

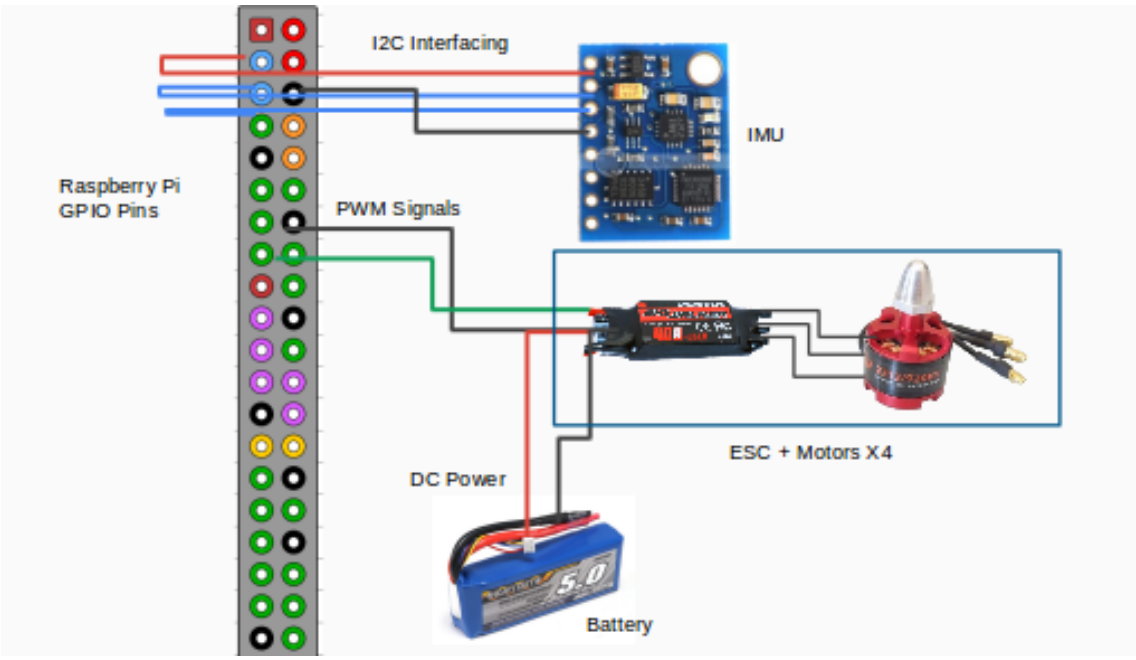

Figure 4.9: Circuit Digram

4.6 FLOW DIAGRAM

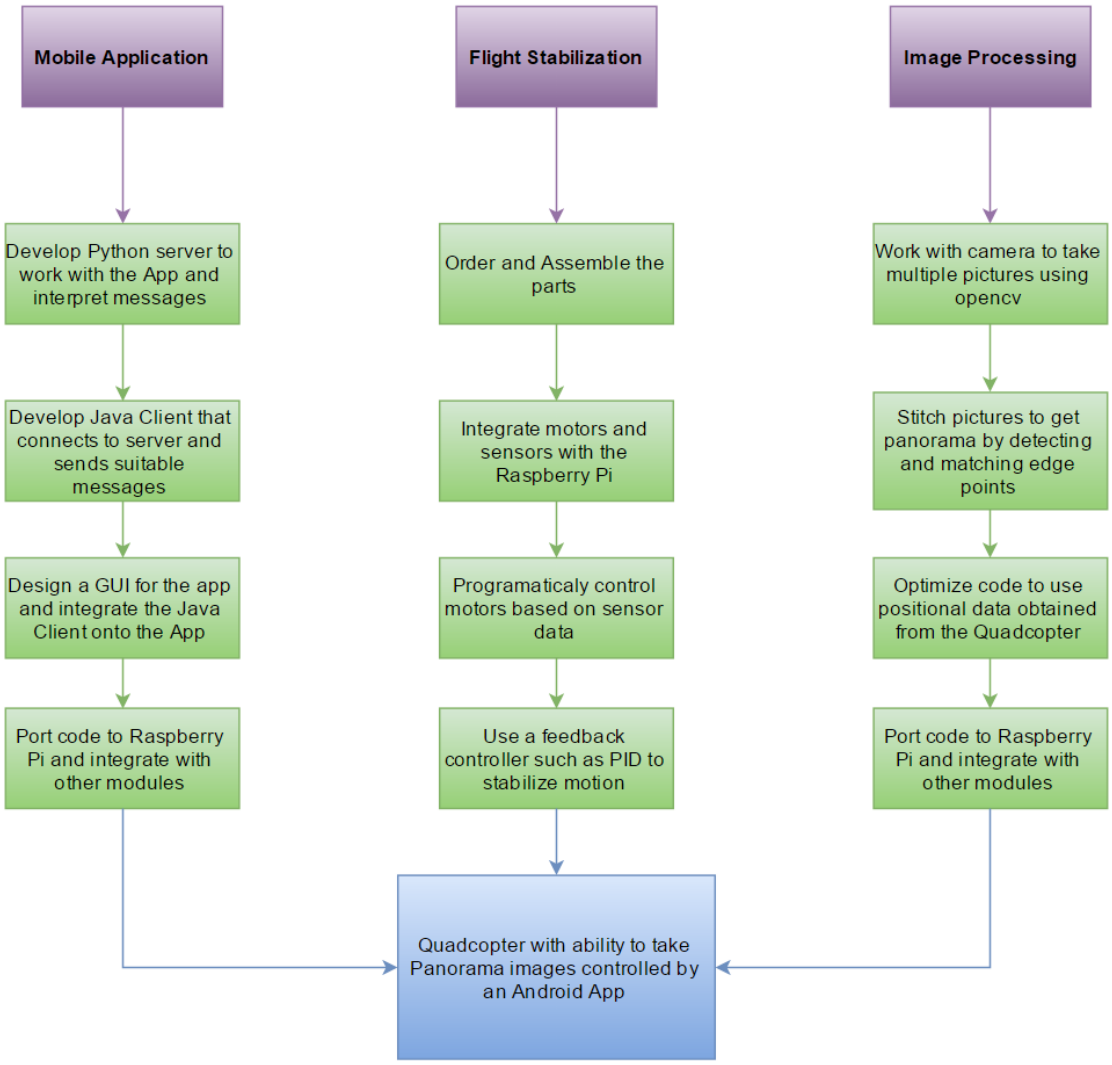

Figure 4.10: Flow Diagram from Input to Output

Description of Flow Diagram

The user begins the quadcopter program by clicking a button on the Android App which is sent as input to the Python server on the Raspberry Pi. These signals are sent over a WiFi network using an Android phone as a mobile hotspot. The user can control the altitude of the drone and the individual motors using the virtual Joystick on the app. Signals are sent as JSON-encoded data to the quadcopter. The orientation data i.e. the roll, pitch and yaw values are detected by the

IMU sensor on the quadcopter and are fed as input to the balancing algorithm. This data is processed by the python program and the orientation of the quadcopter is automatically corrected using the PID algorithm. When the camera signal is triggered, the Pi Camera begins taking pictures at an interval of 1 second and saves them on the SD Card. Once the camera is disabled, the image-processing program is executed on the captured images. The final results i.e. the Panorama image and the reconstructed 3D Point Cloud are separately rendered on a web browser using a remote laptop on the ground. Thus, the quadcopter is semi-autonomous as its flight is controlled by the user but its stability is automatically corrected. It is remotely controlled using an Android app.

CHAPTER 5

SYSTEM DEVELOPMENT

5.1 INPUT AND OUTPUT TO SYSTEM

5.1.1 Input

Input to the system consists of sensor data by the IMU, JSON-encoded orientation data from the app and images taken by the RPi camera to the quadcopter.

5.1.2 Output

The output of the system consists of automatic stabilization of the quadcopter's flight, change in orientation of the quadcoper and image processing functions respectively.

1) An upward swipe on the left joystick results in increase in altitude of the quadcopter

2) A downward swipe on the left joystick results in decrease in altitude of the quadcopter

3) An upward swipe on the right joystick results in the quadcopter pitching forward i.e. increase in the back motors

4) A downward swipe on the right joystick results in the quadcopter pitching backward i.e. increase in the front motors

5) An swipe right on the right joystick results in the quadcopter pitching right i.e. increase in the left motors

6) An swipe left on the right joystick results in the quadcopter pitching left i.e. increase in the right motors

5.2 INPUT AND OUTPUT FOR EACH MODULE

5.2.1 Raspberry Pi-controlled quadcopter

Input

The input to Raspberry Pi is the gesture packets from the internet.

1. IMU sensor-data from the accelerometer and gyroscope regrading acceleration and angular velocity.

2. JSON-encoded orientation data from the Android app.

3. JSON-encoded image-acquisition signal data from the Android App.

Output

There are 3 kinds of outputs based on the input to the quadcopter.

1. IMU sensor data is used ny the PID algorithm to calculate the required orientation and stabilize the quad automatically.

2. Orientation (x,y, z) data is sent to the quad to make it move to the left/right or increase/decrease in altitude.

3. Image acquisition signals allow the RPi camera to click pictures and store them.

Process

Code for Python Server

```
s = socket.socket()
host = '0.0.0.0'
port = 12345
s.bind((host, port))
s.listen(5)
c, addr = s.accept()
print 'Got connection from', addr
bus = smbus.SMBus(1)
 while True:
```

```
  try:
    msg=c.recv(1024)
    data = json.loads(msg)
    if 'reset' in data:
      quadcopter.set_zero_angle()
      continue
    p = int(data['P'])
    i = int(data['I'])
    d = int(data['D'])
    height = int(data['pitch'])
    quadcopter.set_height(height)
    quadcopter.set_PID(p,i,d)
    except Exception as e:
      print e
quadcopter.stop()
c.close()
```

Code for Balancing the Quad using PID

```
while self.running:
          start_time = time.time()
          old_pitch, old_roll, old_yaw = pitch, roll, yaw
          (pitch, roll, yaw) = self.imu.read_pitch_roll_yaw()
          (_, _, gx, gy, _, ax, ay, _) = self.imu.read_all()
          axis_output = {'x': 0, 'y': 0, 'z': 0}
          [axis_output['x'],i_x]=self.compute_PID_output(self.kp_x,
             self.ki_x, self.kd_x, pitch-self.offset_x, i_x,
             old_pitch)
          [axis_output['y'],i_y]=self.compute_PID_output(self.kp_y,
             self.ki_y, self.kd_x, roll-self.offset_y, i_y,
             old_roll)
          [axis_output['z'],i_z]=self.compute_PID_output(self.kp_z,
             self.ki_z, self.kd_x, yaw-self.offset_z, i_z,
             old_yaw)
```

```
self.motor_bl.setW(int(self.height+axis_output['x']/2
  +axis_output['y']/2))
self.motor_br.setW(int(self.height+axis_output['x']/2
  -axis_output['y']/2))
self.motor_fl.setW(int(self.height-axis_output['x']/2
  +axis_output['y']/2))
self.motor_fr.setW(int(self.height-axis_output['x']/2
  -axis_output['y']/2))
end_time = time.time()
print end_time-start_time
while(end_time-start_time <= 0.02):
    end_time = time.time()
    time.sleep(0.0001)
i=i+1
```

5.2.2 Image-Processing Module

This module is split into two parts:

1. Panorama stitching
2. 3D Model Reconstruction

Input - Panorama Stitching

The input consists of images taken by the RPi camera.

Output

Stitched images forming a 360 degree equirectangular panorama image.

Process

```
image_1 = imread(path_to_image_1)
image_2 = imread(path_to_image_2)

(keypoints1, descriptors1) = detect_and_compute(image_1)
(keypoints2, descriptors2) = detect_and_compute(image_2)
matches = knn(descriptors1,descriptors2)
```

```
  points1 = new Array(keypoints1[i] for (i,j) in matches)
  points2 = new Array(keypoints2[j] for (i,j) in matches)
  homography = find_homography(points1,points2)
  (size, offset) = calculate_size(image_1, image_2, homography)
  warped_image = warp_perspective(image_2,homography)
  panorama = warped_image
  panorama[offset_x:offset_x+image_1_width,offset_y:offset_y+image_1_height]
     = image_1
```

Input - 3D Reconstruction

The input consists of images taken by the RPi camera.

Output

Sparse point-cloud representing 3D model of the images taken by the quadcopter rendered on a browser.

Process

```
images = read_images(image_path)
for each image in images:
  features = detect_features(image)
  save_features(features)
matches = {}
for each image1 in images:
  for each image2 in images:
    if image1 not equal to image2:
      count = match_features(image1.features,image1.features)
      if count > threshold:
        matches[image1] = image2
        for each image1 in matches:
          for match in matches[image1]:
            track = add_track(image1,image2)
            create_track_graph(track)
            im1, im2 = find_images_with_largest_matches(images)
            pts3d = bootstrap_reconstruction(im1,im2)
```

```
for each image in images:
  if image not equal to im1 and if image not equal to
     im2:
    pts2d = get_keypoints(im2)
    pts3d = incremental_reconstruction(pts3d,pts2d)
    pts3d = bundle_adjustment(pts3d)
```

5.2.3 Android App Module

Input

Touch and click (Haptic) controlled movements on the virtual joystick interface of the app.

Output

Touch movement on the circular joystick results in change in orientation of the quad. Clicking the camera button send a signal to the RPi Camera to take pictures.

Process

Joystick Interface

```
joystick_left = new Joystick()
joystick_right = new Joystick()
joystick_left.setTouchListener()
joystick_right.setTouchListener()
camera_button.setClickListener()

camera_button.click(){
  camera.shoot()
  camera.enable()
}

while(app_running)
{
  action_left = joystick_left.touch()
  action_right = joystick_right.touch()
```

```
  distance_left = joystick_left.getDistance()
  distance_right = joystick_right.getDistance()
  switch(action_left)
  {
    case 'up': increase_height(distance_left)
               break
    case 'down': decrease_height(distance_left)
               break
    default: print 'Invalid action'
  }
  switch(action_right)
  {
    case 'up': pitch_forward(distance_right)
               break
    case 'down': pitch_backward(distance_right)
                 break
    case 'left': pitch_left(distance_right)
                 break
    case 'right': pitch_right(distance_right)
                  break
    case 'upleft': pitch_upleft(distance_right)
                   break
    case 'upright': pitch_upright(distance_right)
                    break
    case 'downleft': pitch_downleft(distance_right)
                     break
    case 'downright': pitch_downright(distance_right)
                      break
    default: print 'Invalid action'
  }
}
```

5.3 MODULES

5.3.1 Raspberry Pi 2-controlled quadcopter

The objective of the project was to build a semi-autonomous quadcopter capable of self-controlled, stable flight guided via wireless communication through an Android app. A quadcopter, also called a quadrotor helicopter or quadrotor, is a multirotor helicopter that is lifted and propelled by four rotors. Quadcopters differ from conventional helicopters. They use rotors which are able to vary the pitch of their blades dynamically as they move around the rotor hub. In the early days of flight, quadcopters (then referred to as 'quadrotors') were seen as possible solutions to some of the persistent problems in vertical flight; torque-induced control issues (as well as efficiency issues originating from the tail rotor, which generates no useful lift) can be eliminated by counter-rotation and the relatively short blades are much easier to construct. The quadcopter utilises Raspberry Pi microcomputer that runs Python and takes care of all the on-board computation. It also holds an accelerometer, gyroscope, IMU sensor, GPS and a Raspberry Pi native camera. The Raspberry Pi through python programming, interfaces with the IMU to find out the current orientation of the quadcopter and accordingly controls the motors. The IMU consists of an accelerometer, gyroscope (and an optional magnetometer) which accurately provides the current orientation of quadcopter including the G-force and angular velocity. The Raspberry Pi interfaces with the IMU using an I2C interface that allows it to communicate with the accelerometers, gyroscopes and magnetometers independently using only 2 pins(viz. SCL and SDA pins) by using separate addresses for each. The Pi gets the accelerometer and gyroscope angles separately for each axis and computes on them to find out the overall orientation

of the quadcopter. The Raspberry Pi can only supply digital signals that is 0 or 1 i.e. on or off. This does not allow us to directly control the motors to vary their speed. The Electronic Speed Controller (ESC) allows such digital signals to be converted to variable voltage and allow speed regulation of motors by supplying a PWM signal to them. PWM signals are achieved by sending pulses at a certain frequency while controlling the amount of time the pulse is on in a cycle. The longer the pulse is high during a cycle, the higher the voltage supplied to the motor. The ESC also allows an external power souce such as a battery to be used for powering the motors. Our drone uses a 10V LiPo battery. LiPo batteries are commonly used for powering drones because they have high discharge rates required to power the four motors functioning at high speed and requiring order of 20A of current.

The quadcopter's flight is controlled using a PID controller. On an abstract level, The PID algorithm is a closed-loop feedback mechanism that is used in control systems. The controller attempts to minimize the error by adjusting control inputs(P,I,D values). In the quadcopter, the PID controller will be taking data measured by the IMU sensor and comparing that against expected values to alter the speed of the motors to compensate for any differences and maintain balance. The PID controller calculation algorithm involves three separate constant parameters viz. the proportional, the integral and derivative values, denoted P, I, and D. Heuristically, these values can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors and D is a prediction of future errors, based on current rate of change.

The Proportional Gain Coefficient(P) is the most important value. This coefficient determines which value is more important - those measured by the gyroscopes or those defined by human control. The higher

the coefficient, the more sensitive the quadcopter becomes to angular change. If 'P' is too high, the quadcopter will start to oscillate.

The Integral Gain Coefficient(I) is used to increase precision of angular position. This term is extremely useful for combating turbulence due to wind and motors. If this value becomes too high, the reaction speed of the drone will decrease.

The Derivative Gain Coefficient(D) is used to allow the quadcopter to reach the desired altitude more quickly as it amplifies the user input. It is dangerous to increase this value as the quadcopter may become highly susceptible to small changes.

The quadcopter flight and on-board operations are controlled via WiFi using an Android app. It receives signals encoded as JSON objects as steady stream from the App. The signal carries altitude, trims, motor speeds and camera information to the RPi. The signal is received by a Python server running on the RPi which is decoded and fed as input to the Quadcopter program.

The quadcopter also houses a native Pi camera module which has the ability to take high quality photos and videos. It interfaces with the Raspberry Pi using a Camera Serial Interface. The function of the quadcopter is to take pictures and perform two major image-processing operations viz. Panorama Stitching and 3D-Model Reconstruction. The drone can take several images of a wide physical space. The Pi then stitches the images together using OpenCV image-processing library to construct a panorama image (which is a wide photo obtained from multiple images). The second operation is 3D Model reconstruction wherein, the images taken by the quadcopter can be fed to a Structure from Motion (SfM) algorithm to construct sparse point clouds. The quadcopter itself is small enough to control its flight and large enough to hold all the on-board equipment.

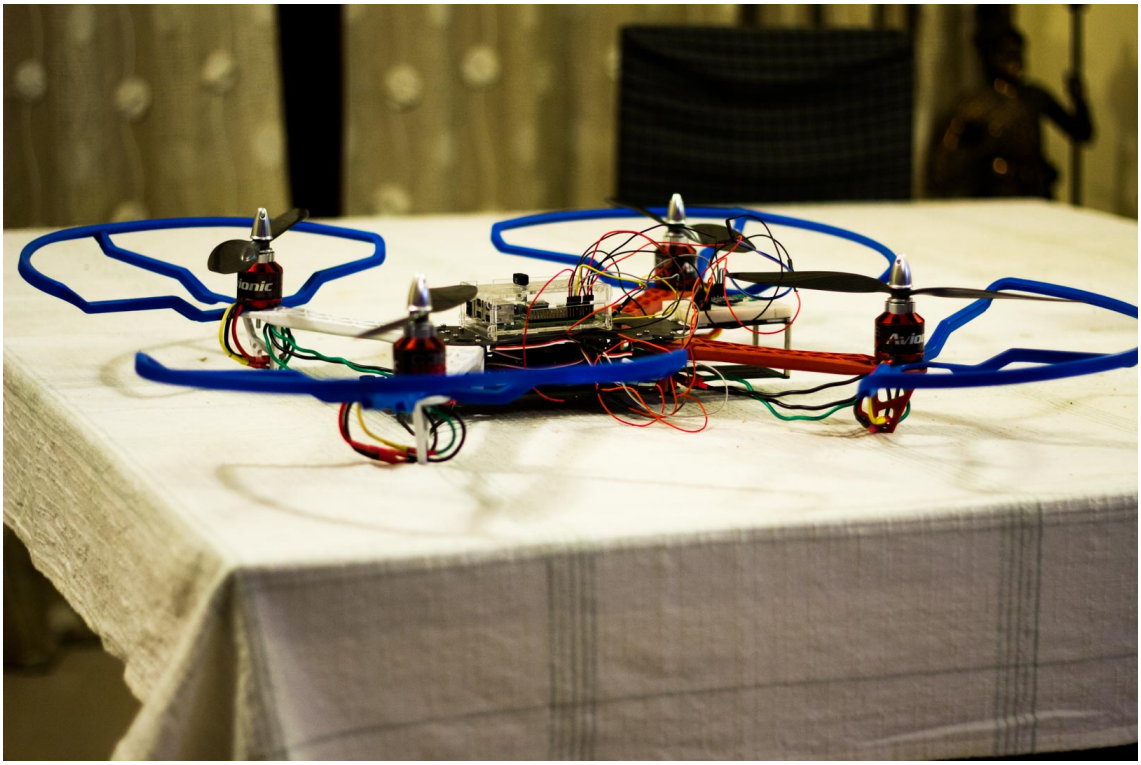

Figure 5.1: The Preliminary Quadcopter without the camera

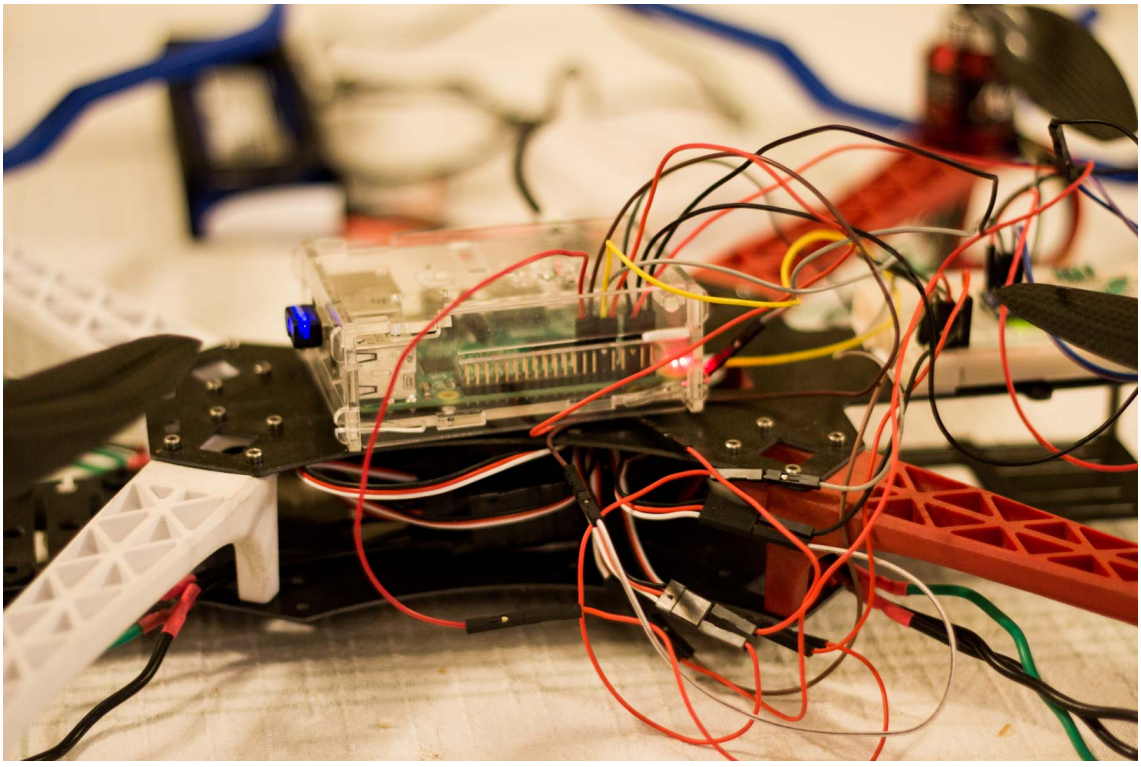

Figure 5.2: Closer look at the wiring of the quadcopter

5.3.2 Image-Processing Module

The objective of the project is to build a quadcopter that can perform Panorama Stitching and 3D Reconstruction of images taken during its flight. The reason we chose these functions is because the drone is mobile and can reach and capture pictures of inaccessible locations like tall buildings and surveying wide-open spaces. OpenCV is an open-source image processing library which has an enormous number of functionally-rich libraries and detailed online documentation. OpenCV Version 3.1.0 (the latest version) is used to provide image-processing support in this project. The image-processing module is divided into two parts Stitching and Reconstruction. The initial stages are the same for both the operations i.e. Acquisition of images, Detecting Features and Matching Features. Feature detection is accomplished using SURF (Speeded Up Robust Features algorithm and Feature Matching is done by building Kd-Trees and using (K-Nearest Neighbours) Flann Algorithm. Feature detection yields keypoints(x,y pixel coordinates and descriptors which accurately describe the region around each pixel). For Panorama Stitching, after feature-matching, RANSAC (Random Sample Consensus) is used to detect outliers and construct a homography matrix. The Homography matrix is essential to the construction of the panorama image as it relates the pixel coordinates in each pair of images. It is then used to warp the images and construct a panorama by aligning them. The stitching process is quite fast for a smaller number images but increases as the number of images increases. In order to speed up the process we have implemented a multi-threaded stitching algorithm that stitches images in parallel, thereby, reducing the time without compromising on the quality(distortion) of the panorama generated. The quality of the panorama is improved if images are taken with large overlapping

features.

3D-Reconstruction is implemented by using Structure from Motion(SfM) algorithm. The preliminary stage is to calibrate the RPi Camera and calculate intrinsic and extrinsic camera paramaters. Once camera calibration is complete, the images have to be preprocessed to collect metadata i.e. EXIF tags which hold important GPS, latitude, longitude and camera information. The next stages are the feature detection and matching stages which have been described above. Matching features are then organized into tracks and a bipartite graph of these ”good tracks is constructed. The next stage is to use the graph to find common tracks and construct the 3D point cloud incrementally. Initially, two images with a large number of matches are chosen for the bootstrap reconstruction. The reconstruction consists of triangulating the 2D points to 3D points which can be visualized as a point cloud. Consequently, the rest of the images' points are added incrementally to the point cloud. After each addition, Bundle Adjustment is done to minimize the reprojection error. This is the most computationally-intensive step and uses the Ceres Solver library. Once all the images have been added to the reconstruction, the final point cloud is generated and stored as a .ply file. The point cloud is visualized as a sparse 3D reconstruction of the object in an interactive browser environment using a Javascript Library called, threeJS on a remote system.

Figure 5.3: Panorama Stitching of test images

Figure 5.4: 3D Reconstruction of test images

5.3.3 Android App

The Android App is used to control all the operations on the quadcopter. It implements a virtual Joystick interface for easy user-interaction. The interface consists of two joysticks, buttons to start the quadcopter program, enable camera mode, dynamically alter x and y trim values and a text box for feeding in the IP address of the RPi.

The two joysticks - left and right are for for altitude control and motor control respectively. The user can swipe upwards and downwards on the left joystick to increase and decrease the height of the quadcopter. Similarly, the user can swipe in all directions on the right joystick to control the speed of individual motors depending on the distance from the center of the joystick.

The x and y trim values can be dynamically changed using the app. These values are used to add bias to the quadcopter and control its stability.

The camera button can enable or disable camera mode on the quadcopter. Enabling it allows the RPi camera to begin shooting pictures with an interval of 1 second while disabling it stops the camera program.

Using an Android app to remotely control UAVs is a level up from traditional RC controllers as it allows users greater flexibility. Users can now control many more aspects of the quadcopter apart from height and orientation such as camera control and bias using a simple and attractive user interface.

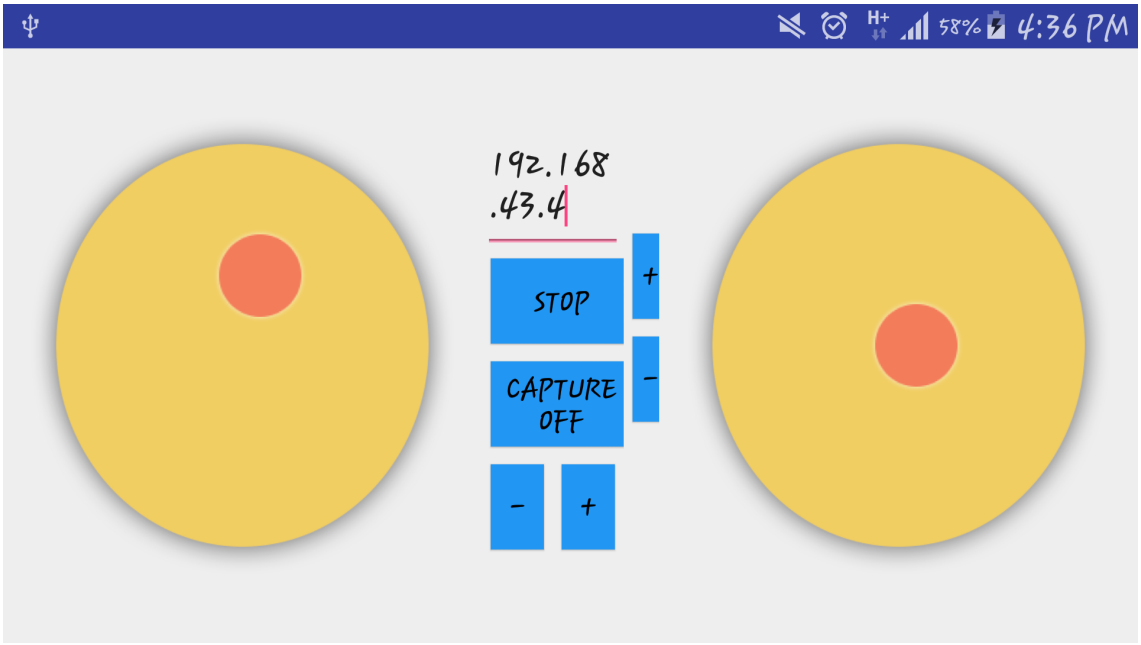

Figure 5.5: Android App Joystick Interface

5.4 OVERALL COMPONENT DIAGRAM

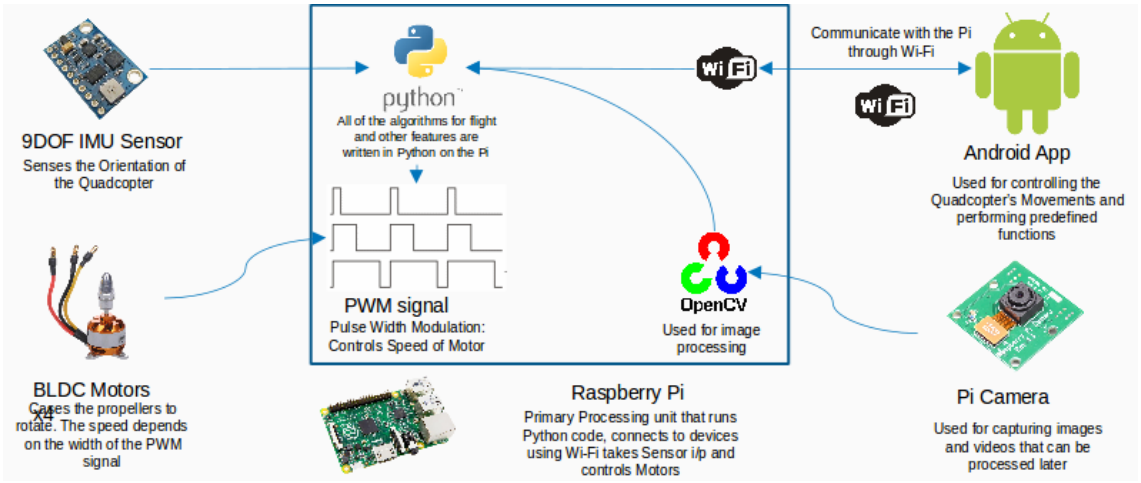

Figure 5.6: Overall Hardware and Software Component Diagram

CHAPTER 6

RESULTS AND DISCUSSION

6.1 RESULTS

6.1.1 Assessment

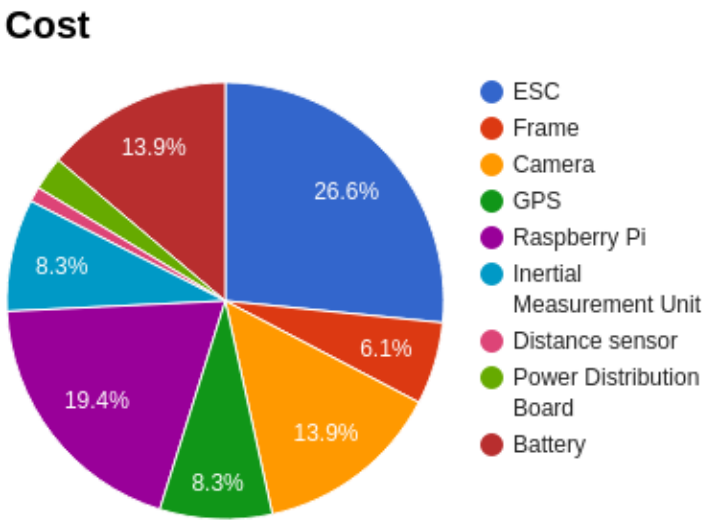

Figure 6.1: Project Cost for each Part

S.No	Signal	Result
1	Swipe left joystick up	Quadcopter flies upward
2	Swipe left joystick down	Quadcopter's height decreases
3	Swipe right joystick up	Quadcopter pitches forward
4	Swipe right joystick down	Quadcopter pitches backward
5	Swipe right joystick left	Quadcopter pitches left
6	Swipe right joystick right	Quadcopter pitches right
7	Click the 'start' camera button	Pi Camera starts shooting
8	Click the 'stop' camera button	Pi Camera stops shooting
9	Click the '+/-' button on the x-axis	Quadcopter's x-trim values change
10	Click the '+/-' button on the y-axis	Quadcopter's y-trim values change

Table 6.1: Test Cases and Results

6.1.2 Evaluation

There are several changes that have to be made when flying the quadcopter in varying environments. The PID values change according to the wind conditions and other disturbances. Although the SD Card is securely inserted into the RPi, due to a large amount of vibration caused in the course of the quadcopter's flight, it may loosen leading to connection loss between the RPi and the App. The quadcopter is controlled by the App from a distance of about 10-15 metres since communication occurs via WiFi. Images are taken by a RPi native camera. The quality of the images taken largely depend on the stability of the quadcopter which are subject to the vagaries of the environment. The pictures taken are stitched and/or reconstructed. This process depends on the number of images taken and the quality of the images.

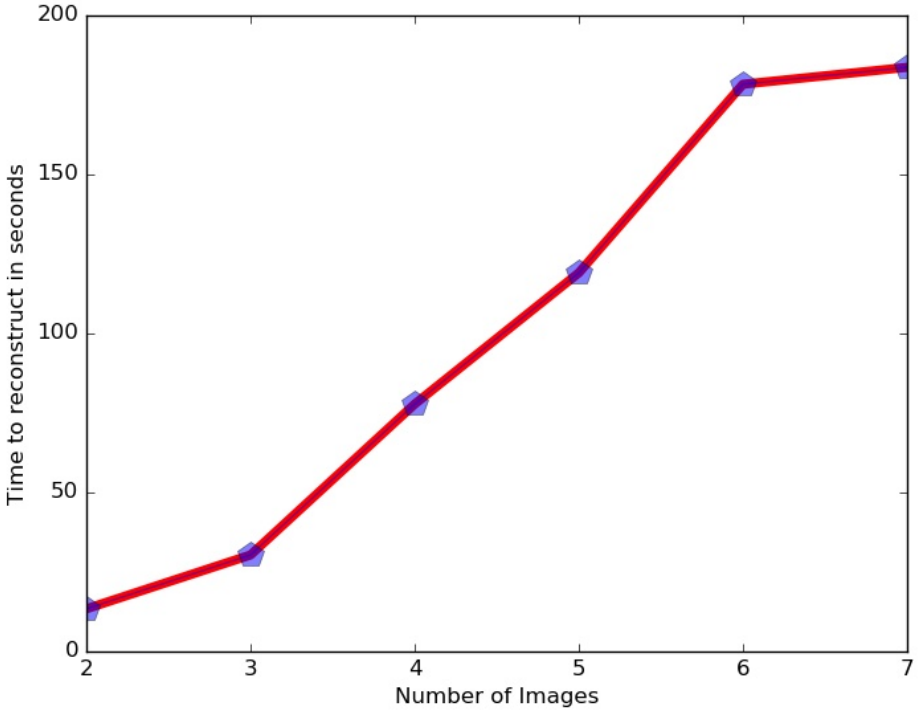

Figure 6.2: Time taken for computation of panorama stitching

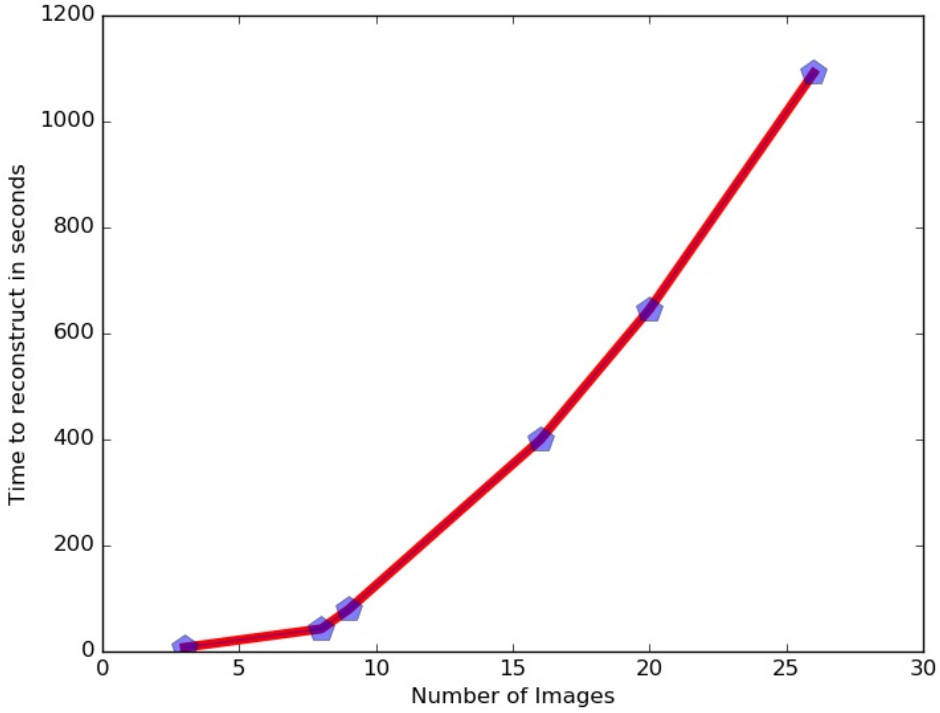

Figure 6.3: Time taken for computation of 3D Reconstruction of images

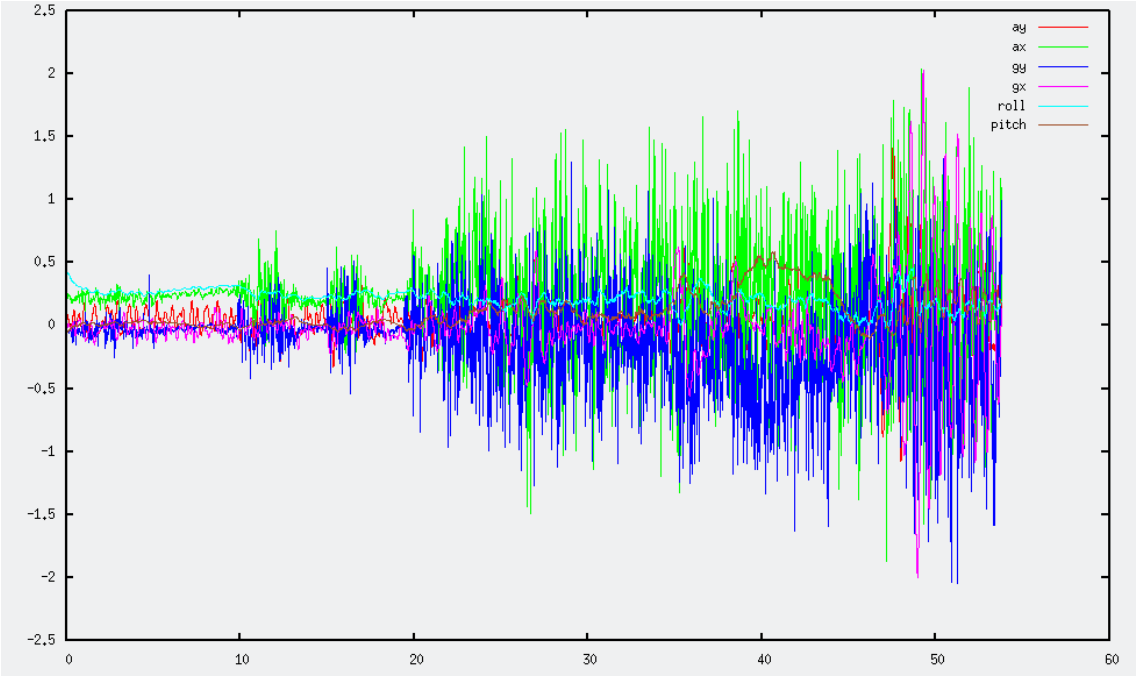

Figure 6.4: Measurement of pitch and roll using complementary filter

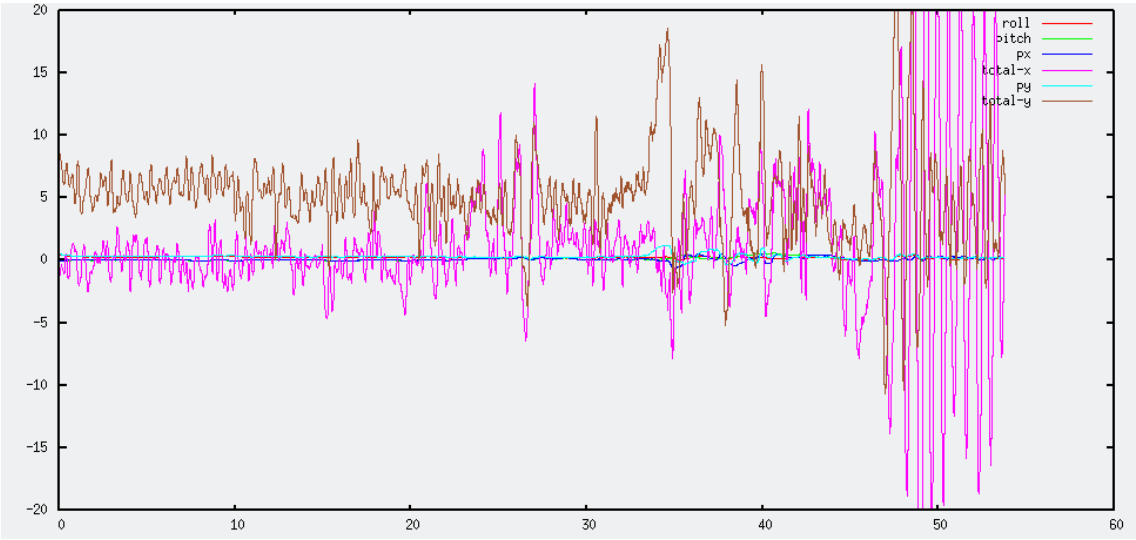

Figure 6.5: Derivation of output using PID controller

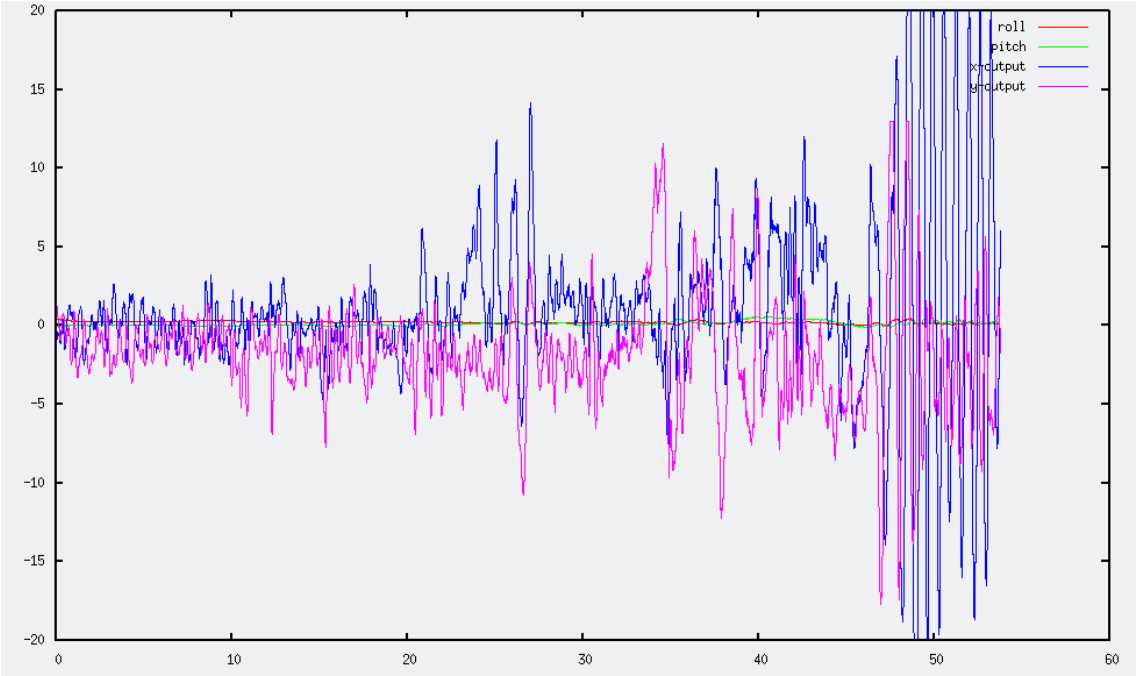

Figure 6.6: Final output with respect to pitch and roll

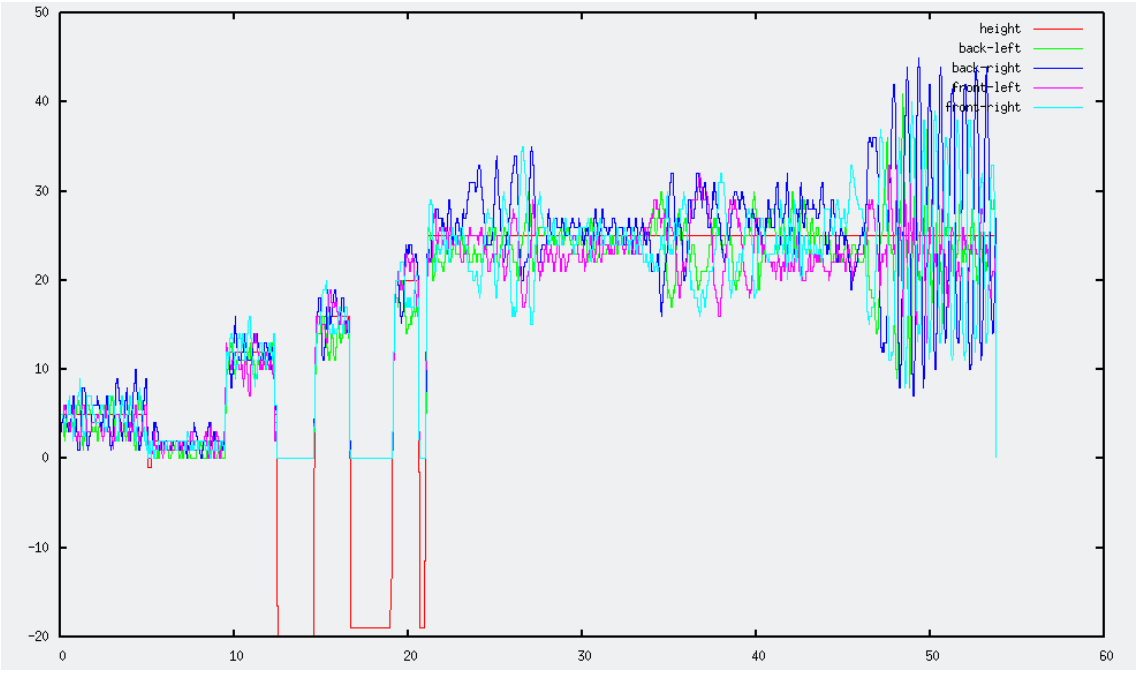

Figure 6.7: Output of motors with respect to height

CHAPTER 7

CONCLUSIONS

7.1 CONTRIBUTIONS

This project establishes a method to integrate the latest advances in aerial robotics and image processing that can ultimately be used for a multitude of applications in domains including aerial photogrammetry, terrain exploration and mapping, archaeological surveys, movie-making and security. The quadcopter combines the two most-often used image-processing functionalities - Panorama and 3D Reconstruction. The extreme mobility of the drone and powerful computing power of the Raspberry Pi form an effective combination for this purpose. The ability to take panaroma pictures allows the quadcopter to reach remote destinations inaccessible by humans and perform a complete panaroma stitching to enable survey of such locations. The ability of the quadcopter to do 3D object reconstruction allows recreation of navigable 3D scenes from locations where the pictures are taken.

7.2 FUTURE WORK

The quadcopter can be made completely autonomous by adding a GPS sensor for navigation and distance sensors or multiple cameras implementing obstacle detection to avoid obstacles. This will enable the user to input the location to the quadcopter where it flies and takes pictures that it will use for 3D reconstruction and panorama stitching.

The quadcopter's RPi camera currently takes pictures all at

once and produces the final Panorama and 3D point cloud. This could be refined to perform the reconstruction and panorama stitching dynamically as pictures are clicked. Various options for panorama such as Polar, Cylindrical etc can be added. With regard to 3D reconstruction, the point cloud generated is a sparse 3D reconstruction of the object. This could be extended to create dense point clouds which would be more realistic. With the integration of these new features, the drone could be used for dynamic terrain mapping, disaster relief, surveillance and tracking.

APPENDIX A

PID Controller

The quadcopter uses a PID controller where 'P' stands for proportional, 'I' stands for integral and 'D' stands for differential. The Proportional term enables the quadcopter to receive a force proportional to the angle of deviation from the central angle as a corrective force to bring it back to its normal orientation. Thus, by giving a high proportional constant, the directive force that will be calculated for a deviation in angle will be large, however setting too large a 'P' value will cause overcompensation resulting in oscillations that will keep increasing.

The Integral term is used to provide a corrective force in cases where the Proportional term cannot compensate for the deviation and angle. In such cases, the deviation and angle is aggregated thereby allowing the quadcopter to get back to its stable state quickly even in cases where turbulence or any external forces are at play. Similar to the Proportional term, too high an integral term will cause oscillations but of a lower frequency then those caused by the Proportional term.

The Differential term is used to provide compensation proportional to that of the change in angle of the quadcopter. Thus, having a high differential term will result in a high corrective force being applied in cases where rapid changes in angle occur. This allows the differential term to either boost or resist the effects of the Proportional and Integral terms because in cases where there are quick angular changes, the Differential term will enable the controller to resist that change and

in cases where the compensation is too fast, the Differential term will correspondingly resist fast changes.

A.1 SAMPLE PID CONTROLLER CODE

```
def compute_PID_output( self, kp, ki, kd, angle, old_i,
   old_angle, limit_p=100, limit_i=100, log=False):
       p = kp * angle
       i = old_i + ki * angle
       d = kd * (angle - old_angle)
       if p > limit_p:
          p = limit_p
       if p < -limit_p:
          p = -limit_p
       if i > limit_i:
          i = limit_i
       if i < -limit_i:
          i = -limit_i
       if log:
          log.write('\t' + str(p) + '\t' + str(i) + '\t' +
             str(d) + '\t')
       return [p + i + d, i]

   def compute_rate_PID_output( self, kpr, kp, ki, kd, gyro,
      angle, old_i, old_angle, limit_p=100, limit_i=100,
      limit_pr=100, log=False):
       p = kp * angle
       i = old_i + ki * angle
       d = kd * (angle - old_angle)
       if p > limit_p:
          p = limit_p
       if p < -limit_p:
          p = -limit_p
       if i > limit_i:
          i = limit_i
```

```
        if i < -limit_i:
            i = -limit_i
        total = kpr*(p + i + gyro)
        print total,p,i,kpr,angle,gyro,kp
        if total > limit_pr:
            total = limit_pr
        if total < -limit_pr:
            total = -limit_pr
        if log:
            log.write('\t' + str(p) + '\t' + str(i) + '\t' +
                str(total) + '\t')
        return [total, i]
```

A.1.1 Sample Joystick Interface Code

```
package com.example.avi.joystick;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.view.ViewGroup.LayoutParams;
public class JoyStickClass {
    private int STICK_ALPHA = 200;
    private int LAYOUT_ALPHA = 200;
    private int OFFSET = 0;

    private Context mContext;
    private ViewGroup mLayout;
    private LayoutParams params;
    private int stick_width, stick_height;
```

```
private int position_x = 0, position_y = 0, min_distance = 0;
private float distance = 0, angle = 0;

private DrawCanvas draw;
private Paint paint;
private Bitmap stick;

private boolean touch_state = false;
private boolean spring_x=false;
private boolean spring_y=false;
public JoyStickClass (Context context, ViewGroup layout, int
   stick_res_id) {
    mContext = context;
    stick =
       BitmapFactory.decodeResource(mContext.getResources(),
           stick_res_id);

    stick_width = stick.getWidth();
    stick_height = stick.getHeight();

    draw = new DrawCanvas(mContext);
    paint = new Paint();
    mLayout = layout;
    params = mLayout.getLayoutParams();
}
public void setSpringy(boolean x,boolean y){
    spring_x = x;
    spring_y = y;
}

public void drawStick(MotionEvent arg1) {
    position_x = (int) (arg1.getX() - (params.width / 2));
    position_y = (int) (arg1.getY() - (params.height / 2));
    distance = (float) Math.sqrt(Math.pow(position_x, 2) +
       Math.pow(position_y, 2));
```

```
angle = (float) cal_angle(position_x, position_y);
if(arg1.getAction() == MotionEvent.ACTION_DOWN) {
    if(distance <= (params.width / 2) - OFFSET) {
        draw.position(arg1.getX(), arg1.getY());
        draw();
        touch_state = true;
    }
} else if(arg1.getAction() == MotionEvent.ACTION_MOVE &&
    touch_state) {
    if (distance <= (params.width / 2) - OFFSET) {
        draw.position(arg1.getX(), arg1.getY());
        draw();
    } else if (distance > (params.width / 2) - OFFSET) {
        float x = (float)
            (Math.cos(Math.toRadians(cal_angle(position_x,
            position_y))) * ((params.width / 2) - OFFSET));
        float y = (float)
            (Math.sin(Math.toRadians(cal_angle(position_x,
            position_y))) * ((params.height / 2) - OFFSET));
        x += (params.width / 2);
        y += (params.height / 2);
        position_x = (int)x - params.width / 2;
        position_y = (int)y - params.height/2;
        draw.position(x, y);
        draw();
    } else {
        mLayout.removeView(draw);
    }
}
 else if(arg1.getAction() == MotionEvent.ACTION_UP) {
    if(spring_x || spring_y) {
        float x = position_x+params.width/2;
        float y = position_y+params.width/2;
        if(spring_x) {
            x = (params.width / 2);
```

```
                position_x=0;
            }
            if(spring_y) {
                position_y=0;
                y = (params.height / 2);
            }
            draw.position(x, y);
            draw();
        }
    }
}

public int[] getPosition() {
    if(distance > min_distance && touch_state) {
        return new int[] { position_x, position_y };
    }
    return new int[] { 0, 0 };
}

public int getX() {
    if(distance > min_distance && touch_state) {
        return (int)((float)position_x*250/getLayoutWidth());
    }
    return 0;
}

public int getY() {
    if(distance > min_distance && touch_state) {
        return (int)((float)position_y*250/getLayoutHeight());
    }
    return 0;
}

public float getAngle() {
    if(distance > min_distance && touch_state) {
```

```
        return angle;
    }
    return 0;
}

public float getDistance() {
    if(distance > min_distance && touch_state) {
        return distance;
    }
    return 0;
}

public void setMinimumDistance(int minDistance) {
    min_distance = minDistance;
}

public int getMinimumDistance() {return min_distance;}
public void setOffset(int offset) {OFFSET = offset;}

public int getOffset() {
    return OFFSET;
}

public void setStickAlpha(int alpha) {
    STICK_ALPHA = alpha;
    paint.setAlpha(alpha);
}

public int getStickAlpha() {
    return STICK_ALPHA;
}

public void setLayoutAlpha(int alpha) {
    LAYOUT_ALPHA = alpha;
    mLayout.getBackground().setAlpha(alpha);
```

```
}

public int getLayoutAlpha() {
    return LAYOUT_ALPHA;
}

public void setStickSize(int width, int height) {
    stick = Bitmap.createScaledBitmap(stick, width, height,
       false);
    stick_width = stick.getWidth();
    stick_height = stick.getHeight();
}

public void setStickWidth(int width) {
    stick = Bitmap.createScaledBitmap(stick, width,
       stick_height, false);
    stick_width = stick.getWidth();
}

public void setStickHeight(int height) {
    stick = Bitmap.createScaledBitmap(stick, stick_width,
       height, false);
    stick_height = stick.getHeight();
}

public int getStickWidth() {return stick_width; }

public int getStickHeight() {
    return stick_height;
}

public void setLayoutSize(int width, int height) {
    params.width = width;
    params.height = height;
}
```

```
public int getLayoutWidth() {
    return params.width;
}
public int getLayoutHeight() {
    return params.height;
}

private double cal_angle(float x, float y) {
    if(x >= 0 && y >= 0)
        return Math.toDegrees(Math.atan(y / x));
    else if(x < 0 && y >= 0)
        return Math.toDegrees(Math.atan(y / x)) + 180;
    else if(x < 0 && y < 0)
        return Math.toDegrees(Math.atan(y / x)) + 180;
    else if(x >= 0 && y < 0)
        return Math.toDegrees(Math.atan(y / x)) + 360;
    return 0;
}

private void draw() {
    try {
        mLayout.removeView(draw);
    } catch (Exception e) { }
    mLayout.addView(draw);
}

private class DrawCanvas extends View{
    float x, y;
    private DrawCanvas(Context mContext) {
         super(mContext);
     }
     public void onDraw(Canvas canvas) {
         canvas.drawBitmap(stick, x, y, paint);
     }
     private void position(float pos_x, float pos_y) {
```

```
            x = pos_x - (stick_width / 2);
            y = pos_y - (stick_height / 2);
        }
    }
}
```

A.2 SAMPLE PANORAMA STITCHING CODE

```
import cv2, numpy as np
import math
import argparse as ap
import time
from threading import Thread

def extract_features(image1,image2, surfThreshold=1000,
   algorithm='SURF'):
  try:
    image_gs1 = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
  except TypeError:
    return
  try:
    image_gs2 = cv2.cvtColor(image2,cv2.COLOR_BGR2GRAY)
  except TypeError:
    return
  detector = cv2.xfeatures2d.SURF_create()
  (keypoints1,descriptors1) =
     detector.detectAndCompute(image_gs1,None)
  (keypoints2,descriptors2) =
     detector.detectAndCompute(image_gs2,None)
  keypoints1,keypoints2 = np.float32([kp.pt for kp in
     keypoints1]), np.float32([kp.pt for kp in keypoints2])
  return (keypoints1, descriptors1,keypoints2, descriptors2)

def find_correspondences(keypoints1, descriptors1, keypoints2,
   descriptors2):
```

```
  matches = match_flann(descriptors1, descriptors2)
  points1 = np.float32([keypoints1[i] for (_, i) in matches])
  points2 = np.float32([keypoints2[i] for (i, _) in matches])
  return (points1, points2)

def calculate_size(size_image1, size_image2, homography):
  (h1, w1) = size_image1[:2]
  (h2, w2) = size_image2[:2]
  top_left = np.dot(homography,np.asarray([0,0,1]))
  top_right = np.dot(homography,np.asarray([w2,0,1]))
  bottom_left = np.dot(homography,np.asarray([0,h2,1]))
  bottom_right = np.dot(homography,np.asarray([w2,h2,1]))
  top_left = top_left/top_left[2]
  top_right = top_right/top_right[2]
  bottom_left = bottom_left/bottom_left[2]
  bottom_right = bottom_right/bottom_right[2]
  pano_left = int(min(top_left[0], bottom_left[0], 0))
  pano_right = int(max(top_right[0], bottom_right[0], w1))
  W = pano_right - pano_left
  pano_top = int(min(top_left[1], top_right[1], 0))
  pano_bottom = int(max(bottom_left[1], bottom_right[1], h1))
  H = pano_bottom - pano_top
  size = (W, H)
  X = int(min(top_left[0], bottom_left[0], 0))
  Y = int(min(top_left[1], top_right[1], 0))
  offset = (-X, -Y)
  return (size, offset)

def merge_images(image1, image2, homography, size, offset,
    keypoints):
  (h1, w1) = image1.shape[:2]
  (h2, w2) = image2.shape[:2]
  panorama = np.zeros((size[1], size[0], 3), np.uint8)
  (ox, oy) = offset
  translation = np.matrix([
```

```
    [1.0, 0.0, ox],
    [0, 1.0, oy],
    [0.0, 0.0, 1.0]
  ])
  homography = translation * homography
  cv2.warpPerspective(image2, homography, size, panorama)
  panorama[oy:h1+oy, ox:ox+w1] = image1
  height, width = panorama.shape[:2]
  crop_h = int(0.05 * height)
  crop_w = int(0.015 * width)
  panorama = panorama[crop_h:-crop_h, crop_w:-crop_w]
  panorama = panorama[int(oy*0.7):,:]
  return panorama

def match_flann(des1, des2,ratio=0.75):
    matcher = cv2.DescriptorMatcher_create('BruteForce')
    rawMatches = matcher.knnMatch(des1, des2, 2)
    matches = []
    for m in rawMatches:
      if len(m) == 2 and m[0].distance < m[1].distance * ratio:
        matches.append((m[0].trainIdx, m[0].queryIdx))
    return matches

def draw_correspondences(image1, image2, points1, points2):
  (h1, w1) = image1.shape[:2]
  (h2, w2) = image2.shape[:2]
  image = np.zeros((max(h1, h2), w1 + w2, 3), np.uint8)
  image[:h1, :w1] = image1
  image[:h2, w1:w1+w2] = image2
  for (x1, y1), (x2, y2) in zip(np.int32(points1),
     np.int32(points2)):
    cv2.line(image, (x1, y1), (x2+w1, y2), (2555, 0, 255),
       lineType=cv2.LINE_AA)
  return image
```

```
def pano(images,i):
  if (i+1) <= (len(images)-1):
    try:
      image1 = cv2.imread(images[i])
    except TypeError:
      image1 = images[i]
    try:
      image2 = cv2.imread(images[i+1])
    except TypeError:
      image2 = images[i+1]
  else:
    return
  (keypoints1, descriptors1,keypoints2, descriptors2) =
     extract_features(image1,image2)
  (points1, points2) = find_correspondences(keypoints1,
     descriptors1, keypoints2, descriptors2)
  correspondences = draw_correspondences(image1, image2, points1,
     points2)
  cv2.imwrite('pano_dataset/door/correspondences.jpg',
     correspondences)
  try:
    (homography, _) =
       cv2.findHomography(points2,points1,cv2.RANSAC,4)
  except Exception:
    print 'Not enough matches!'
    return -1
  (size, offset) = calculate_size(image1.shape, image2.shape,
     homography)
  images[i] = merge_images(image1, image2, homography, size,
     offset, (points1, points2))
  if(len(images) == 2):#final panorama
    filename = 'pano_dataset/door/pano_ multi_final'+str(i)+'.jpg'
  else:
    filename = 'pano_dataset/door/pano_multi'+str(i)+'.jpg'
  cv2.imwrite(filename,images[i])
```

```
if __name__ == '__main__':
  import time
  st = time.time()
  images =
     ['pano_dataset/door/door1.jpg','pano_dataset/door/door2.jpg','pano_dataset
  n = len(images)
  val = int(math.ceil(math.log(len(images),2)))+1
  for q in range(val):
    for i in xrange(0,len(images),1):
          print 'i: ',i,' len: ',len(images)
          t = Thread(target=pano, args=(images,i))
          t.start()
          t.join()
          if i+1 <= len(images)-1:
            del images[i+1]
```