<u>QR Code Detection Landing System, Start Here Document</u>

Fall 2022

Andy Zheng, Archit Kalla, Grant Eichenberger, Daniel Carvalho, Kunal Rastogi

File / Component: **Inland Raspberry Pi Camera v1.3**

<u>Description/Instructions:</u>

For this project we chose an Inland Raspberry Pi Camera v1.3. It was relatively easy to obtain and inexpensive, so it could be used in the future, or be replaced with a camera with a better field of view. This camera has dimensions of 24 by 25 nanometers, a focal length of 3.29 millimeters, and a FOV of 65 degrees. Our camera is used to capture an image of a QR code and send this data to the Pi so it can reorient the UAV to be positioned on top of that code. The camera is connected to a Raspberry Pi through a ribbon cable, and mounted on the test UAV using a protruding piece of laser cut wood.

File / Component: **Protruding Mount**

<u>Description/Instruction:</u>

The current mount option that we have integrated onto the test UAV is a piece of laser cut wood with a hole cut for the camera lens at one end. The Pi Camera v1.3 is laid flat on the top, with the ribbon running along the piece of wood into the Pi 3B. This piece of wood protrudes from the front facing side of the UAV, so as to not have its field of view blocked by any of the cables or other components. In the future, this could be replaced with a 3D printed mount possibly on the underside of the drone.

File / Component: **QR_LAND Folder**

<u>Description/Instruction:</u>

This folder contains the necessary files needed to achieve movement to a QR landing pad. The files descriptions in this folder are as follows:

- <u>QR_Landing.py</u>: This file contains the main ROS function that attempts to land the drone on QR code. **The file here differs from what is present on the Pi as we were unable to copy the file from the Pi and put it in the folder, so the Pi has the most up-to-date "working" version of the landing code**. **The code should be in ~/catkin_ws/src/QR_LAND/scripts/QR_Landing.py.** This is written in python 2.7. The code leverages pyzbar to identify the qr code on in the image. Then using the bounding box from this package we find a center point and determine how far in x and y coordinates the qr code is in terms of the center of the camera. We then move the drone to those coordinates. Important to note that these coordinates that are published are absolute coordinates in the real world, not relative to the drone. **We made an oversight in the code that needs to be corrected for proper landing: when publishing the coordinates**

we forgot to add the qr relative position to the drones current position. The fix for this is when publishing qr_pos do the following:

```
#publish position
last_xy = ( feedback_pos.pose.position.x + qr_pos[0], feedback_pos.pose.position.y +
qr_pos[1])
pose.pose.position.x = last_xy[0]
pose.pose.position.y = last_xy[1]
```

This should fix the issue where the drone would land incorrectly.
If a QR code is not detected, the drone should move in a spiral formation; the code for this is present on the Pi version. We recommend referring to that instead of the one in this folder

- <u>Examine_area.py</u>: An empty file, we were unable to implement traversing the space in an effective manner, QR_landing.py has a temporary solution to just move in a spiral to land on the Pi. In reality we would want this file to include code to handle cases where partial qr detection occurs, right now that is not handled in the pyzbar package.
- <u>Test_qr_detection.py</u>: self explanatory, tests for qr code in the frame of the camera and displays bounding boxes.
- <u>Get_calib_images.py:</u> Gets calibration images for undistortion of the camera for more precise movements. We were unable to undistort so this is not needed for QR_land to function. Only needs to be done once
- <u>Get_camera_distortion.py:</u> gets the distortion matrix and saves it as a .npz file. This file is then used in the undistort() function in qr_landing.py to undistort the images, but we chose not to undistort the image as it cost performance. If you choose to use this the B.npz must be present in the same folder as QR_landing in the ros folder **~/catkin_ws/src/QR_LAND/scripts/** so that the undistort() function works properly. This file only needs to be run once to get the camera lens distortion.

To run the code on the drone simply input this into the ssh terminal on the drone "rosrun QR_LAND QR_landing.py".

File / Component: **DEPTH_BASED_LANDING Folder**
<u>Description/Instruction:</u>
This contains files that were basically half implementations of depth based landing requiring the stereo cameras. This code is written in Python 3. The depth_calculation.py contains code to calculate depth of the entire image. Then there is an unfinished function to find the depth of a patch. In landing_pad_identify.py there is a file that finds the edges and vertices of the largest contour in the image, in our case the landing pad and then finds its vertices. The goal was to find the depth of each vertice and determine if each vertice of the landing pad was within 7 cm of each other to determine pitch and roll. We were trying to keep within 10 degrees pitch or roll so

if needed to take-off again after landing it could do so safely. There is code also for undistortion and calibration of camera, although this has not been tested as we abandoned depth based landing.

File / Component: **OFFBOARD_CONTROL Folder**
Description/Instruction:
This folder contains the necessary files needed to achieve offboard (autonomous) control of the drone using MAVROS/QGroundControl.

- px4_v1.13-main
  This compressed folder contains the version of firmware for the pixhawk that we had to update to in order for offboard control to work. Later versions of the firmware may work as well however our scripts were run using this version
- The firmware can be uploaded to the pixhawk board using the steps found here: https://docs.qgroundcontrol.com/master/en/SetupView/Firmware.html and selecting the “Custom Firmware File…” option under step 1.
- The control script itself is included as a part of the QR_LAND routine.

File / Component: **LANDING_PAD_FILES Folder**
Description/Instruction:
This folder contains files for laser cutting landing pads for the UAV, which a QR code can be etched onto. The files contain different sizes of landing pads for testing convenience.

Resources:

- PX4 User Guide: https://docs.px4.io/master/en/
  - Good information about the PX4 flight stack.
- Autodesk Inventor: https://academy.autodesk.com/software/inventor
  - Good information about how to work with Inventor.
- The **Final Report** in the folder gives a more detailed description of the current state of the project and what would be needed to build on it.

Contacts:
Project Advisors: Professor Nikolaos Papanikolopoulos
Travis Henderson

Hardware/Software: Grant Eichenberger
Archit Kalla
Andy Z
Kunal Rastogi
Daniel Carvalho - carva037 University of Minnesota