Mobile Robotics Lab Spring 2015
Automatic Landing on a Moving Target

Chen Minhua, Guo Jiangtao

July 1, 2015

1 Introduction

Autonomous landing for Unmanned Aerial Vehicle (UAV) is an important aspect in the field of UAV research. If you want to build a autopilot plane, you have to let it use automatic landing method to land itself on particular target. Automatic landing can also be used in forced landing when a plane come up with some problem like running low on battery. it can also be applied in military helicopters which need to landing in some harsh environment. Currently, there are some relative autonomous landing methods such as GPS (Global Position System) and INS (Inertial Navigation System). However, if the landing environment is very bad, these methods won't be robust enough. As computer vision is developing in a high speed, some approaches based on computer vision have been widely applied in UAV automatic landing research. We can also combine these methods together to get better performance.

In addition, nowadays, the inexpensive quadricopter toys becoming widely available and usually these toys also provide better sensing features and not bad computational power. In our project, we plan to use the AR.Drone, made by the Parrot Inc,to land on some special marker on the ground or some moving plane. The AR.Drone is a high-tech low cost flying toy with powerful hardware and sensors including 1GHz 32 bit ARM Cortex A8 processor with 800MHz and also video DSP, 1GB DDR2 RAM at 200MHz two cameras(one of the camera is 720p HD Camera), WIFI bgn, 3 axis gyroscope, accelerometer, and magnetometer, pressure sensor, ultrasound sensors and many other properties[1].

2 State of Art

In Allen C. Tsai, Peter W. Gibbens and R. Hugh Stone's paper[7], they present a method to estimate the full 3D position information of a UAV by integrating visual cues from one single images with data from an Inertial Measurement Unit (IMU) under the Kalman Filter formulation. They use two or more frames of image data with feature enriched information in the image to estimate the

[1]for more details about the AR.Drone, you can find the website here.

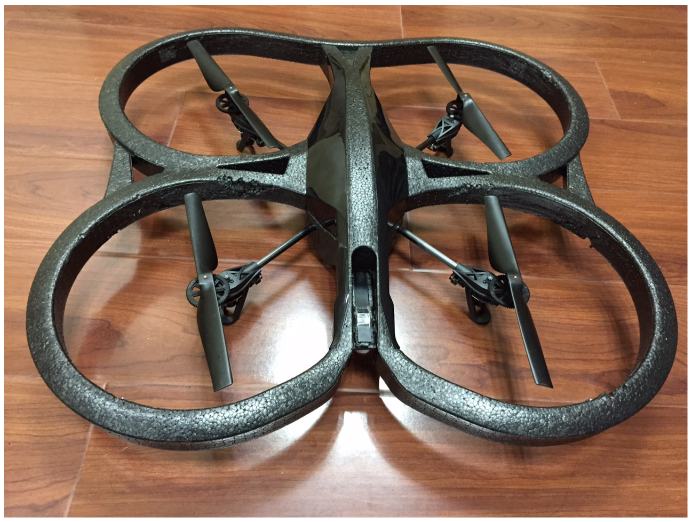


Figure 1: The AR.Drone 2.0 in our project, with the front camera modified to toward down

3D position of the UAV. They use a rather conventional type of landing pad with visual features extracted for use in the Kalman filter to obtain optimal 3D position estimates.

In Daquan Tang, Fei Li, Ning Shen, Shaojun Guo's paper[6], they present a vision-based method for attitude and position estimation, based a few feature points distributed on the ground arbitrarily. In their methods, calculating the attitude and position relative to runway is the core. Firstly, the coordinates in the camera frame of the feature points was acquired with the N-point algorithm; secondly, with the orthogonalization method, the rotation matrix and the translation vector between the camera frame and the runway frame were acquired. They also used least-median-squares (LMS) algorithm to diminish the influence of noise and to improve the robustness.

In Patrick Benavidez, Josue Lambert, Aldo Jaimes and Mo Jamshide 's papers [1] and [2], they present control system for an UAV which provides aerial support for an unmanned ground vehicle(UGV), the UGV acts as a mobile launching pad for the UAV. the UAV provides additional environmental image feedback to the UGV. They also use the Parrot AR.Drone2.0 quadcopter as the UAV hardware because of "its agile flight and video feedback capabilities". Their paper presents design and simulation of the fuzzy logic controllers for performing landing, hovering, and altitude control. Image processing and Mamdani-type inference, which is the most commonly seen fuzzy methodology proposed in 1975 by Ebrahim Mamdani, are used for converting sensor input into control signals used to control the UAV.

In Nick Dijkshoorn's paper[4] and his master thesis[3], he presents a real-time SLAM approach for affordable MAVs with a down-looking camera. And the approach has been validated with the AR.Drone quadrotor helicopter. His development is partly based on simulation which requires both a realistic sensor and motion model. His thesis describe how a visual map of the environment can be made. The visual map consists of a texture map used for human navigation and a feature map used by the AR.Drone to localize itself. He presents a novel approach to robustly recover the translation and rotation between a camera frame and the map. An experimental method to create an elevation map with a single airborne ultrasound sensor is presented in the thesis. His experiments validated that his presented methods work in a variety of environments. Furthermore, the impact of the camera resolution and various pose recovery approaches are investigated in his thesis.

3 Approach

Described in the introduction of above, we know that what we want to do is to control the unmanned aerial vehicle(UAV) stably and land the UAV to some special target panel. For example the target panel has some special markers on it, such as QR code, or the Augmented Reality markers(AR_tag)[2]. In this project, we want to consider the automatic landing problem and find an efficient and useful methods for aircraft. We use AR.Drone as the air platform and try to land it on a fixed and moving target. Firstly, we need to guarantee that the AR.Drone can recognize the landing target correctly. We use the AR tag to achieve this function. Secondly, We should let the AR.Drone try to get to the desired place and hover stably above the landing target. Then, according to the information we get from the down look camera in the AR.Drone image(we have made some hardware change to let the original forward look camera on AR.Drone 2.0 look down to get better image resolution), we can calculate the relative coordinates with respect to the landing platform and move the AR.Drone above the target stably. PID controller is used to keep AR.Drone stable moving. Finally, we can land the AR.Drone on the target smoothly and stably.

The hardware in our project:

- AD.Drone 2.0
- Laptop with Ubuntu 14.04 and ROS indigo
- Turtlebot(if available, can be used as the moving target. In our project, not used yet.)

The software in our project:

- ROS indigo
- Ubuntu14.04

[2]For more detail about the Tag, here is the website, ATT LAB

- AD.Drone 2.0 onboard drivers
- AR_tag ros package `ar_track_alvar`
- ros package `ardrone_autonomy` and `ardrone autonomy Lab`, AR.Drone 2.0 driver ros wrapper
- ros package `tum_ardrone`.a very nice AR.Drone project, and give us much help, such as the PID controller, EKF, odometry estimation.

3.1 Camera calibration

Before everything starts, the first thing is to calibrate the camera. In following parts, camera info will be used in the marker recognition package. So, we refer to the documents of AR.Drone driver `ardrone autonomy` and ros camera calibration. Finally, ros camera calibration tools `Camera Calibrator`[3] are used in our project.

Here is the command we have used in our project.

```
$ rosrun camera_calibration cameracalibrator.py --size 8x6
--square 0.0245 image:=/camera/image_raw camera:=/camera
```

After running this tools, the camera calibration matrix will be got, Here are our calibration data and has been put into the ros `ardrone_front.yaml` file in the folder `/.ros/camera_info/`.

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [569.883158064802, 0, 331.403348466206, 0,
568.007065238522, 135.879365106014, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.526629354780687, 0.274357114262035,
0.0211426202132638,-0.0063942451330052, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
```

[3]for more detail, you can find on the ros wiki here

```
  data: [463.275726318359, 0, 328.456687172518, 0,
0, 535.977355957031,134.693732992726, 0, 0, 0, 1, 0]
```

Then, we set up the whole system and do some test about the distance estimation. In table1, show us the test results. In the table, we can see that, the row of origin should be the origin point `(0,0,0)`, but we can see not all be so, there are really errors. And we also find that the roll, pitch and yaw of the AR.Drone are not exactly be the right value for this, we have try best to let the three angle be the right value. Such as if we put the AR.Drone horizontal and the roll and pitch should be about $-180°$ or about 0. However, we try our best to change the position of AR.Drone to make the value right but failed. Finally, we find that these errors are the system errors of the AR.Drone sensors, And later, the AR.Drone can fly back and be under control with these errors. But we should notice that the distance is different in test 1, test 2 and test 3. The left down and right down corner is the card corner in test 3 because of the small value of the distance rather than that in test 1 and 2 the corner is marker corner, as shown in figures below the whole paper.

3.2 Marker recognition

Before the drone can land on the target, the first thing is to localize itself with respect to the environment in the AR_tag frame. So, AR markers are used to implement an indoor localization system with respect to the known markers in our experiment.

The AR markers are augmented reality markers which are characterized by different patterns of black and white squares arranged in grid. Such markers are well suited for pattern recognition approaches and can easily be detected with some known software, such as the ALVAR Tracking Library[4] which is a C++ software library for creating virtual and augmented reality application and is maintained by the VTT ALVAR Tracking Library. In addition, for this library, there is a Robot Operating System(ROS) wrapper for it, which is called ar_track_alvar package[5]. In this package, it has 4 functionalities, generating AR tag of varying size, resolution, and data/ID encoding, identifying and tracking the pose of the individual AR tags, identifying and tracking the pose of "bundles" consisting of multiple tags, and using camera images to automatically calculate spatial relationship between tags in a bundle.

In our project, with the help of the ROS package `ar_track_alvar`, we can get the `Pose` of AR tag including the position (x, y, z) and the rotation angle denoted by the quaternions of AR tag wrt. camera frame. The points in the AR tag frame and camera frame respectively denoted as ${}^{A}\mathbf{P}$ and ${}^{C}\mathbf{P}$, So we get

[4]the official website about the library:here
[5]the website is here

Table 1: Camera calibration test for the height and distance test results, in the table, the position is w.r.t. marker in camera frame, the every test situation is below the whole report.The AR tag marker diameter is 0.132 meters, so the abs coordinates of four corner is all 0.66 meters. And the "big left down " and the "big right down" is left and right down corner of the big card. We have show this situation clearly in the below figures.

Height	Test 1 0.81m	Test 2 0.51m	Test 3 1.21m
pose	(x,y,z,roll, pitch, yaw)		
units	(m,m,m, $^\circ$,$^\circ$,$^\circ$)		
origin	(0.02,0.02,0.81) (-176.95,1.52,0.83)	(0.00,-0.00,0.51) (-174.55,0.09,-0.45)	(0.03,0.04,1.21) (-176.89,1.15,-0.04)
corner 0(left down)	(-0.08,-0.11,0.81) (-172.01,-1.80,-0.14)	(-0.06,-0.09,0.51) (-172.72,-0.27,0.05)	(-0.06,-0.30,1.21) (-171.22,3.61,-0.13)
corner 1(left up)	(0.04,0.05,0.81) (-174.20,5.97,-0.00)	(-0.05,0.03,0.51) (-171.21,0.41,0.04)	(0.04,0.12,1.21) (-178.32,3.36,0.03)
corner 2(right down)	(0.12,-0.10,0.81) (-172.91,2.83,-0.08)	(0.08,-0.07,0.51) (-174.46,0.92,-0.18)	(0.24,-0.34,1.20) (-170.03,6.17,0.12)
corner 3(right up)	(0.14,0.05,0.81) (-174.07,4.67,0.03)	(0.10,0.05,0.51) (-173.73,2.78,0.25)	(0.22,0.01,1.21) (-173.45,5.78,-0.03)
error estimation	(m,m)	(m,m)	(m,m)
corner 0(left down)	(0.02, 0.02)	(0,0)	(0.03, 0.04) ("big left down")
corner 1(left up)	(0.02, 0.05)	(0,0.03)	(0.08,0.07)
corner 2(right down)	(0.06, 0.04)	(0.02, 0.01)	(0.1, 0.11) ("big right down")
corner 3(right up)	(0.08, 0.01)	(0.04, 0.01)	(0.16, 0.05)

$$\begin{aligned} {}^{A}\mathbf{P} &= {}^{A}_{C}\mathbf{T}\,{}^{C}\mathbf{P} \\ &= {}^{A}_{C}\mathbf{T}^{-1}\,{}^{C}\mathbf{P} \\ &= {}^{C}_{A}\mathbf{T}\,{}^{C}\mathbf{P} \end{aligned} \tag{1}$$

Where ${}^{A}_{C}\mathbf{T}$ and ${}^{C}_{A}\mathbf{T}$ represent respectively the transform from AR tag to camera frame and from camera frame to AR tag frame. And, from the ROS package `ar_track_alvar`, we get the point position wrt. camera frame, so we can easily derive the position wrt. AR tag frame from the above Eq1. In the code, thanks to the ROS transform tools, these trivial but important things has been done efficiently.

Here, for comparison purpose, we also use the QR code to provide the information about the helipad to the UAV. QR codes have become common in consumer advertising and every usual life. The QR code will give the actual size of the the QR code and also the number about the helipad to distinguish between other helipad. Also, we will get the QR code from the rgb camera below the AR.Drone. With the help of the Zbar library in Linux, we can get the information in the QR code.

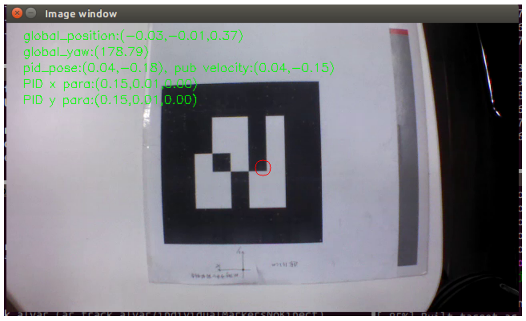
Comparing AR tag with QR code, we can know that, the QR code can give us more information about itself like the diameter about itself. When we use the AR tag, we can just get the QR code diameter from the information resolved from the image processing rather than set the fixed diameter of the QR code before using it. This is a better aspect than the AR tag. However, in the image processing, we find that the Zbar library processes and give out the information of QR code has much more delay than the AR tag processing. Thus we also know from the ros AR tag package that the recognition time cost will not dramatically cumulate with the AR tag number increasing. So one of the AR tag usage is that put a bundle of AR tag on an object, when the robot just get the information of parts of the whole bundle of AR tag, it can also recognize the object and do some action.

3.3 UAV and moving target control

In our project, we use the AR.Drone as the UAV. Actually the images from the down-looking camera(some hardware hacks have been done) of the AR.Drone have been used. Once we get the frames of the image, let it transmit the images via WiFi to the control Laptop located on the ground. Of course, based on these image frames, the control node will analysis these frames, do the landing target localization on the Laptop, and send the commands back to the drone. And also, we know that we can use the ROS (robot operating system) to operate the UAV. With the help of ROS, we can easily control the UAV. In the ROS wiki, we find the `tum_ardrone` package. This package builds on the well known monocular SLAM framework PTAM presented by Klein & Murray in their paper[5].

In detail, based on the every a few milliseconds received image frame, we set up a controller to calculate the distance between the AR tag center and the image center along x and y coordinates. Next, we use the PID controller to keep more stable velocity control. Then publish the control command to the AR.Drone for moving the AR.Drone close to the image center. The target we have used in our experiments are the AR tag marker on the board. The actual marker in the experiment has show in the figure 2.

(a) ardrone track the marker



```
global_position:(-0.03,-0.01,0.37)
global_yaw:(178.79)
pid_pose:(0.04,-0.18), pub velocity:(0.04,-0.15)
PID x para:(0.15,0.01,0.00)
PID y para:(0.15,0.01,0.00)
```

(b) AR tag recognition window

Figure 2: AR tag in experiments

3.4 Stable UAV

In our previous project, we find it's usually unstable in the control system of AR.Drone, so we should make some improvement for the stable control of drone. Here, we know the most common used stable controller is proportional integral derivative controller (PID controller), which is a control loop feedback mechanism widely used in industrial control system. A PID controller calculates an error value as the difference between a measured process variable (in our project, it is the horizontal distance between the camera center and the AR tag center) and a desired set point (the position what we want)[6]. In the velocity control progress, we use the PID controller to make velocity control more accurate and stable.

In our project, we set the formula as

$$U_t = K_P(X_{des} - X_{t-1}) + K_D(\dot{X}_{des} - \dot{X}_{t-1}) + K_I \int_0^t (X_{des} - X_{t'-1})dt' \quad (2)$$

where the U_t, X_{t-1}, X_{des} are respectively out put of PID controller, the position at time $t-1$, and the desired position which set to $(0,0)$ in our experiments. The $\dot{X}$ denotes the respectively velocity at different time. the K_P, K_I, K_D respectively represent the proportional integral derivative parameters which should be determined in the actual usage and depends on the system. In our system, because we want to move the drone just above the AR tag, so here we set the desired position as $(0,0)$. we just apply the PID controller along x,y coordinates to control the v_x, v_y velocity.

[6]for more detail refer to wikipedia PID_controller

3.5 The whole system

The overview of our system is as follows:

- First, we let the UAV search and detect the static target with the markers (AR_tag or QR code) on the ground from the every frame of the rgb camera.
- Second, let the UAV move slowly to the above of the target and follow it tightly and stably, with the help of height from the rgb image based on the ROS package `ar_track_alvar`.
- Then, when the UAV locks the target, keeping locking the target, let the UAV move down slowly until to some close height such as 0.5 metres in our experiments.
- Final, correct the target marker in the center of the image and move down stably and slowly until landing on the helipad.

The whole system structure is shown in the figure 4a. In this flow chart, we can get this control sequence.

- First start and test the keyboard control node in the our project package. we have provide this node to control the drone with keyboard. The commands are as follow:

  ```
  i,k,j,l: for forward, backward, left, right,
  value = 0.3
  q,a: for up, down, value = 0.4
  blank space: for stop and hover.
  h: for hover
  t,d: for takeoff and land
  ```

- Second, drone get the image frame from camera, image processing, no marker, hover and search the marker. if yes, go ahead.
- Third, drone get the marker, do the calculation to get the position (x, y, z) wrt. AR tag frame and the distance $(\Delta x, \Delta y)$ between the image center and AR tag center.
- Forth, make decision about how to do next step, if close enough, publish command to move the drone down. if no, move the drone closer.
- Fifth, we should get the height of drone again, when moving down, the drone should check the height incessantly to decide what to do next step. if the height lower than the threshold (0.5 meters, you can set your own value), land the drone. if not, back to step 2.

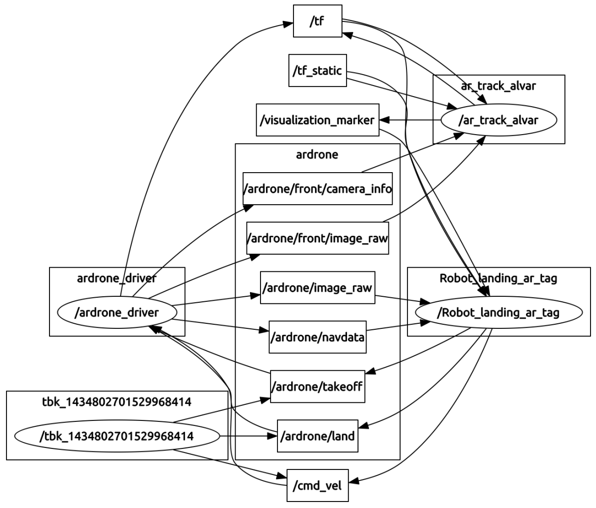
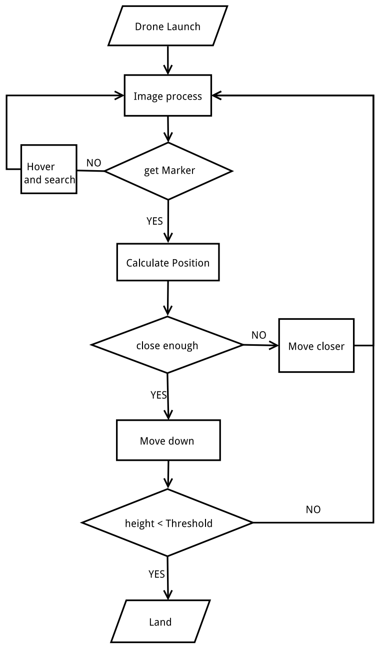
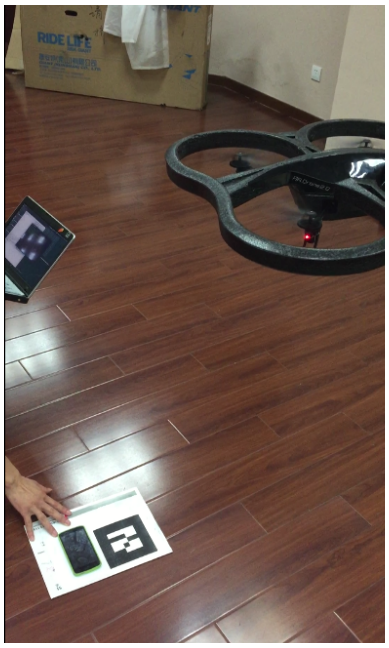


Figure 3: The whole system topic and node relationship





(a) The whole system structure

(b) The AR.Drone follow marker in experiment

Figure 4: AR tag in experiments

In addition, the figure 3 shows the relationship between our topic and the drone driver. The node `Robot_landing_ar_tag` is our main node, the node `tbk_1434802701529968414` is the keyboard control node in our project. From the node relationship, we can see that our node use the image node `/ardrone/image_raw` for image information and data node `/ardrone/navdata` for sensors data. And our main node publish command to the topic `/cmd_vel` to control the drone. The node `/ardrone_driver` and `/ar_track_alvar` is the ROS drone driver package `ardrone_autonomy` node and the AR tag recognition package `ar_track_alvar`.

4 Experiments and Results

What we want to do is to land the UAV to the moving target. So we do some experiments to test our result. In our experiments, about the moving target, we move the marker on board manually rather than use the moving turtlebot.

- The first thing is to recognize the target. We hold the UAV to turn around to search the target marker(the AR_tag and the QR code), recognize the marker on the fixed ground and give us the information about the marker. So we can compare the distance from our code calculation and the real value measured with tape.
- Let the UAV use the information to calculate the coordinate of target to prepare for landing.
- Test the stable hovering of the drone with the PID control. For this purpose, let the UAV flying above the fixed marker lasting for 10 minutes, compare the input velocity and the measured velocity from the ROS AR.Drone driver topic `navdata`, calculate the errors.
- Test the UAV flying above the fixed target stably, we should manually measure the error between the camera center and the image center of the marker.
- Test landing on the fixed target slowly and stably.
- Test the searching target process of drone.
- Test the searching and landing process of drone.
- Compare the effect on the flying stability of the drone of different markers, AR_tag and QR code, flying stability of drone with two kinds of markers will be compared. The above tests all should be tested and measured.

Above is what we want to do the test and finally we do test them. For example the figure 2a and figure 4b are the picture of drone in the experiments. Finally, we can let the drone hover on the marker, follow the marker moving stably, and land on the marker. The detailed results what we get are here:

- We compare the results from the main node, we find the distance estimation has not little errors. this results have been show in the table 1.
- Our node can achieve the marker recognition quickly and stably, and also give us the position of the camera w.r.t. AR tag marker.
- The PID controller do improve the stability of the drone movement. And we finally give the parameters are in the code.
- The drone can last for a long time hovering above the marker and don't lose the marker.
- When moving the marker manually, our node can follow the marker moving stably, but also has some little wobble.
- finally, the drone can recognize the marker and follow the marker, then land on the marker slowly.

Comparing the AR tag marker and QR code, we find that the AR tag recognition is fast, thus in our project, it's better to use the AR tag than the QR code. And about the AR tag, we can also try to use bundles of AR tags to get more wide sense. However, it also has some drawbacks like the information in one AR tag is just the ID about itself rather than that strings in the QR code. And in our experiments, AR tag recognition has better performance including marker recognition, fast calculation than QR code. The QR code are better than AR tag in some aspects including more information about target. However, the distance calculation between the image center and the marker center and the height estimation still need continue to improve.

5 Future Development

Above all, we have achieve the basic project target including marker recognition, marker above hovering, mobile marker following, and landing drone on the mobile marker. And also, we again verify the assumption of marker based navigation of UAV landing. However, what we done in our project still has much to improve.

- For the stability of drone control, PID controller has been used in our project. But we still has some work to do about the accurate controller such as try some other fuzzy logical controller.
- Because of the draft of the drone and the errors in the sensors, we try to use the marker as the visual navigation "sensors". But if the "visual sensors" still has much errors, then the controller still does wrong thing and sends right command. So the next thing must be done is to improve the visual feedback data accuracy. Such as improving the QR code height calculation method and improving the AR tag marker height estimation with our own algorithm.

- In our project, the visual feedback just has a limit ability to feedback, because of the view limitation of cameras. if the camera lost the marker, then the drone just has simple searching strategy or just hover there. So in this situation, the visual feedback will fail. And this problem is really problem especially when the drone moves down and down, the problem will be more and more apparent. In our project, we dose not calculate the global position of the drone. So it's much necessary for us to find some method to do estimation of global position. Here the traditional way is to do the global position estimation from the odometry. In our project, This also can be done to improve the performance.

- Finally, for global position of drone, we also can try to use some other sensors or features. The former is that kinect can be used for the global position estimation of moving target in the air. The later is that the monocular SLAM(simultaneous localization and mapping) algorithms can be used to locate the position of drone.

6 Time-line

2015/05/05–2015/05/16 Do some research, paper reading, and test the ROS AR_tag library. Implement and Test the PID controller of the drone control.

2015/05/17–2015/03/30 Again read and be familiar with the wiki about ROS, AR.Drone driver. Recognize the markers, Test and calculate the 3D coordinate of the target marker in the global frame.

2015/05/30–2015/06/09 Test the prototype of the project and Do improvement.

2015/06/10–2015/06/19 Set up the whole system, do the final test, measure the result, and write slides.

2015/06/20–2015/06/25 Do more test, finish the final experiments and report.

7 Acknowledge

Firstly, we will thank professor Sören Schwertfeger for giving us much help in our project, giving us a very nice introduction to the mobile robot, and providing us with so nice hardware AR.Drone 2.0, turtlebot, and kinect(ASUS Xtion pro). Secondly, I will thank to my partner ChenMinhua, without his help, i can not finish the whole project. Thirdly, We will thank to our classmates ChenXiao and ZhangChi. Thanks them for accompanying with us to studying such interesting. Finally, We will also thank our University for providing us so nice place to study and do experiments.

References

[1] Patrick Benavidez, Josue Lambert, Aldo Jaimes, and Mo Jamshidi. Landing of an ardrone 2.0 quadcopter on a mobile base using fuzzy logic. In *World Automation Congress (WAC), 2014*, pages 803–812. IEEE, 2014.

[2] Patrick J Benavidez, Josue Lambert, Aldo Jaimes, and Mo Jamshidi. Landing of a quadcopter on a mobile base using fuzzy logic. In *Advance Trends in Soft Computing*, pages 429–437. Springer, 2014.

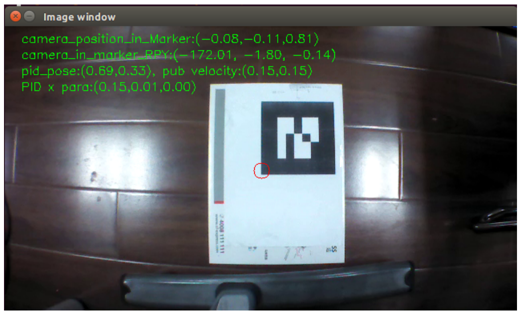
[3] Nick Dijkshoorn. Simultaneous localization and mapping with the ar. drone. *PhD diss., Masters thesis, Universiteit van Amsterdam*, 2012.

[4] Nick Dijkshoorn and Arnoud Visser. Integrating sensor and motion models to localize an autonomous ar. drone. *International Journal of Micro Air Vehicles*, 3(4):183–200, 2011.

[5] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

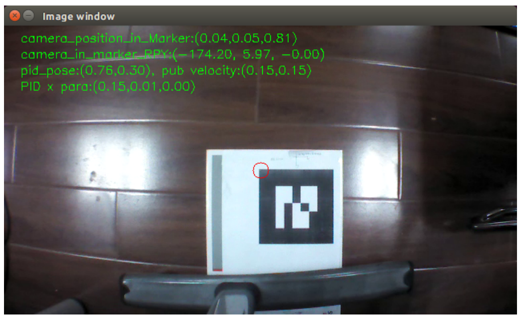
[6] Daquan Tang, Fei Li, Ning Shen, and Shaojun Guo. Uav attitude and position estimation for vision-based landing. In *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on*, volume 9, pages 4446–4450. IEEE, 2011.

[7] Allen C Tsai, Peter W Gibbens, and R Hugh Stone. Terminal phase visual position estimation for a tail-sitting vertical takeoff and landing uav via a kalman filter. In *Optics East 2007*, pages 67640P–67640P. International Society for Optics and Photonics, 2007.



```
camera_position_in_Marker:(−0.08,−0.11,0.81)
camera_in_marker_RPY:(−172.01, −1.80, −0.14)
pid_pose:(0.69,0.33), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

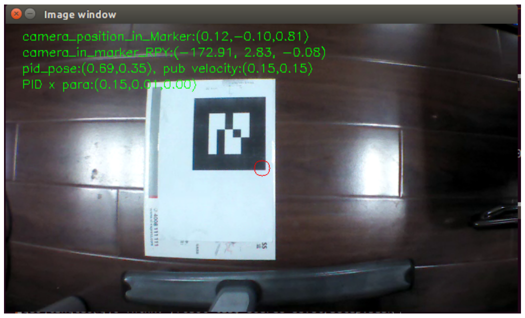
(a) test1 left down



```
camera_position_in_Marker:(0.04,0.05,0.81)
camera_in_marker_RPY:(−174.20, 5.97, −0.00)
pid_pose:(0.76,0.30), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

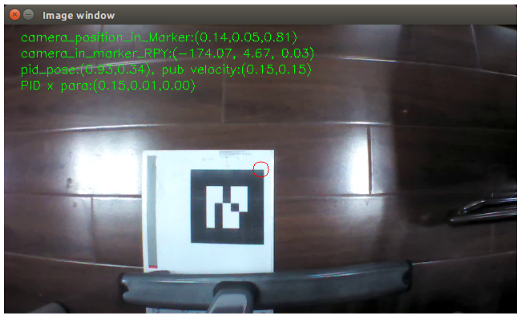
(b) test1 left up

Figure 5: Markers



```
camera_position_in_Marker:(0.12,−0.10,0.81)
camera_in_marker_RPY:(−172.91, 2.83, −0.08)
pid_pose:(0.69,0.35), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

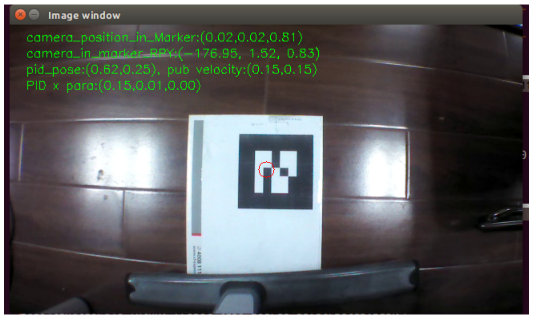
(a) test1 right down



```
camera_position_in_Marker:(0.14,0.05,0.81)
camera_in_marker_RPY:(−174.07, 4.67, 0.03)
pid_pose:(0.93,0.34), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

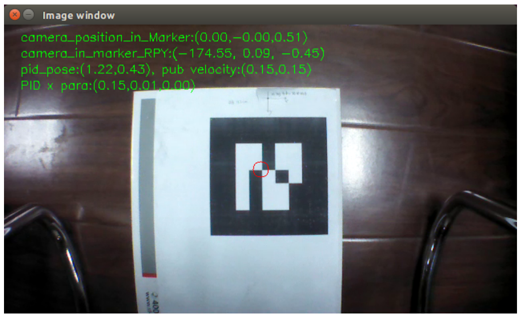
(b) test1 right up

Figure 6: Markers



```
camera_position_in_Marker:(0.02,0.02,0.81)
camera_in_marker_RPY:(−176.95, 1.52, 0.83)
pid_pose:(0.62,0.25), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

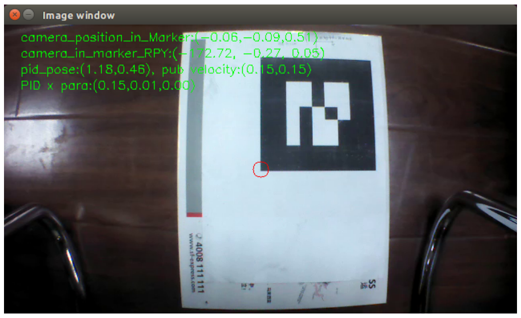
(a) test1 origin



```
camera_position_in_Marker:(0.00,−0.00,0.51)
camera_in_marker_RPY:(−174.55, 0.09, −0.45)
pid_pose:(1.22,0.43), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

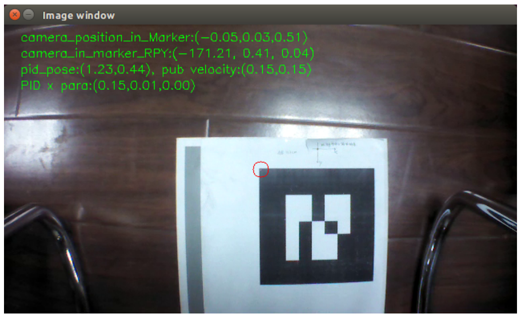
(b) test2 origin

Figure 7: Markers



```
camera_position_in_Marker:(−0.06,−0.09,0.51)
camera_in_marker_RPY:(−172.72, −0.27, 0.05)
pid_pose:(1.18,0.46), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

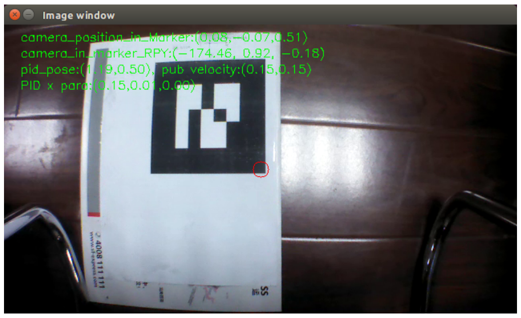
(a) test2 left down



```
camera_position_in_Marker:(−0.05,0.03,0.51)
camera_in_marker_RPY:(−171.21, 0.41, 0.04)
pid_pose:(1.23,0.44), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

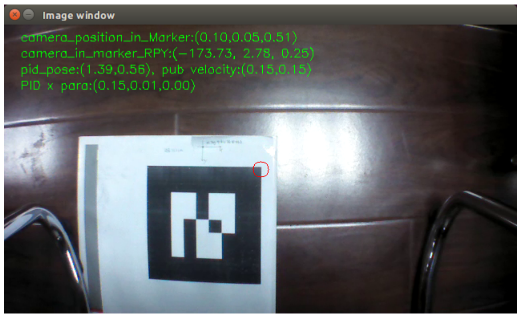
(b) test2 left up

Figure 8: Markers



```
camera_position_in_Marker:(0.08,-0.07,0.51)
camera_in_marker_RPY:(-174.46, 0.92, -0.18)
pid_pose:(1.19,0.50), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

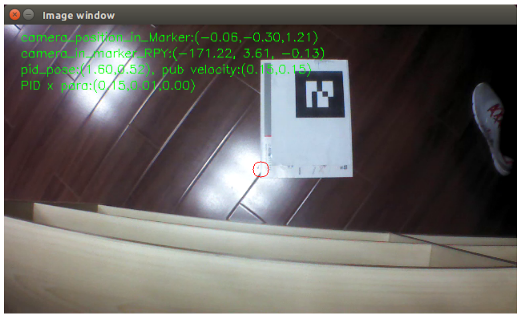
(a) test2 right down



```
camera_position_in_Marker:(0.10,0.05,0.51)
camera_in_marker_RPY:(-173.73, 2.78, 0.25)
pid_pose:(1.39,0.56), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

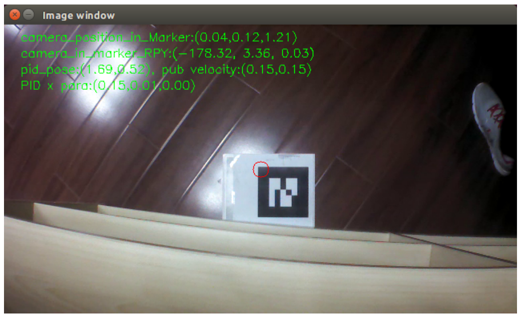
(b) test2 right up

Figure 9: Markers



```
camera_position_in_Marker:(-0.06,-0.30,1.21)
camera_in_marker_RPY:(-171.22, 3.61, -0.13)
pid_pose:(1.60,0.52), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

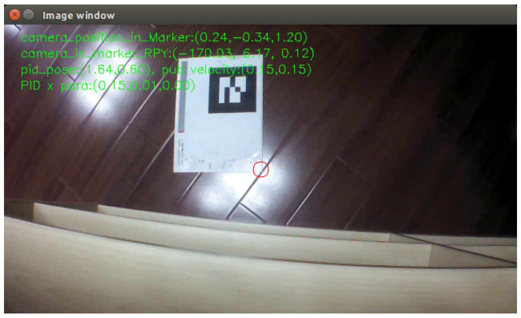
(a) test3 left down



```
camera_position_in_Marker:(0.04,0.12,1.21)
camera_in_marker_RPY:(-178.32, 3.36, 0.03)
pid_pose:(1.69,0.52), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

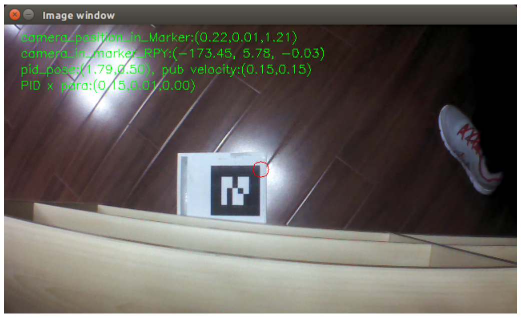
(b) test3 left up

Figure 10: Markers



```
camera_position_in_Marker:(0.24,-0.34,1.20)
camera_in_marker_RPY:(-170.03, 6.17, 0.12)
pid_pose:(1.64,0.60), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

(a) test3 right down



```
camera_position_in_Marker:(0.22,0.01,1.21)
camera_in_marker_RPY:(-173.45, 5.78, -0.03)
pid_pose:(1.79,0.50), pub velocity:(0.15,0.15)
PID x para:(0.15,0.01,0.00)
```

(b) test3 right up

Figure 11: Markers