Cooperative Robotics

Cooperative Tracking and Landing of Unmanned Areal Vehicle on a Mobile Platform

Dipendra Subedi[1] and Emanuele Sansebastiano[2]

European Master on Advanced Robotics Plus (EMARO+)
Jaume I University, Spain
[1]*star.dipendras@gmail.com*, [2]*emanuele.sansebastiano@outlook.com*

December 2016

Abstract

Recent years have seen the use of miniature unmanned aerial vehicles (UAVs) become more popular for search & rescue and delivery missions, but tracking and landing on moving targets is still a major hurdle in their usage. This project presents a modular system comprising of a micro UAV and a mobile platform in which the UAV is capable of performing trajectory tracking of the mobile platform and is able to land on it. The Parrot AR drone 2.0 is used as the UAV and the Turtlebot as the mobile platform. Control architecture is established for the data-exchange and cooperation between UAV and mobile platform. The in-built vision based tracking module of Parrot is used for the tracking of tag placed on the top of Turtlebot. Circular, linear and randomly generated trajectories were used for evaluation purposes. The quadcopter was able to successfully track and land on the mobile platform with small errors allowing for safe indoor flights.

1 Introduction

Aerial robotics is a rapidly growing area in the field of robotics with quadcopters playing a central role in its research activities (Hoffmann, Huang, Waslander, & Tomlin, 2007). A quadcopter consists of four independently controlled motors which govern how it flies. The real potential of a quadcopter lies in its civilian and military applications (Sarris & Atlas, 2001). Due to its hovering capabilities together with high maneuverability, it can fly at low altitudes without collision with the environment making it a popular choice for various tasks such as aerial photography, delivery drones and search & rescue missions.

One of the key challenges faced in the aerial robotics field is the problem of landing the drone on a remote platform. Although significant work has been done with stationary platforms, the prospect of landing on a mobile platform is greatly underdeveloped. The task can be broken down into two subtasks, namely the tracking and the landing.

Most commercial drones rely on GPS as the main position feedback (Kim, Jung, Lee, & Shim, 2014) but their accuracy (approximately 3 m) (Kleusberg & Langley, 1990) is not up to the mark for high precision demanding applications such as the ones discussed above. Although the use of vision based

algorithms for localization is becoming popular but the low payload capacities of the micro unmanned aerial vehicles (μUAV) such as quadcopters limit the size of the onboard computer and ultimately the performance of the algorithm suffers (Majumder & Shankar, 2014). There exists external position feedback systems which provide high accuracy localization but they limit the autonomy of the drone.

The tracking problem has been addressed by many in recent years with the traditional proportional-integral-derivative (PID) controller being a popular choice among the researchers. Bouabdallah et al (Bouabdallah, Noth, & Siegwart, 2004) demonstrated its use and successfully landed the quadcopter on a 20cmx20cm area. But his approach is only effective when the tracking object has a smooth trajectory. Wenzel et al (Wenzel, Masselli, & Zell, 2011) applied a similar approach but the success rate for landing on smaller targets was not good. Another popular approach for target tracking is the use of an extended kalman filter (EKF) like the one used by Prevost et al (Prevost, Desbiens, & Gagnon, 2007) but this requires the dynamics of the system to be accurately represented in the observer system.

There are various approaches that can be used to perform the landing. Different approaches based on missile guidance laws have been established by Gautam et al (Gautam, Sujit, & Saripalli, 2015) with pure proportional navigation being most effective for landing a drone.

The objective of this project was to develop a control software architecture that handles the data-exchange between UAV and mobile platform. Once a suitable architecture has been established, it is expanded to accommodate a high-level control module which handles the mobile platform tracking and the quadcopter landing tasks. Robotic operating system (ROS) has been used as the framework.

section 2 describes the complete system used for this work. The proposed control software architecture is explained in the section 3. The section 4 describes the experimental results and analysis. The conclusions and the future works are mentioned in the sections 5 and 6 respectively.

2 System Description

The complete system used for this work comprises of: the UAV, `Parrot AR Drone 2.0` (fig.1(a)); and the mobile platform, `Turtlebot` (fig.1(b)).

(a) Parrot AR Drone 2.0 (b) TurtleBot 2

Figure 1: UAV and Mobile Platform

2.1 Parrot AR Drone 2.0

The Parrot AR-Drone is an electrically powered quadcopter intended for augmented reality games. It consists of a carbon-fiber support structure, plastic body, four high-efficiency brushless motors, sensor and control board, two cameras and indoor and outdoor removable hulls. The control board not only ensures safety by instantly locking the propellers in case of a foreign body contact, but also assists the user with difficult maneuvers such as takeoff and landing. The drone operator can set directly its yaw, pitch, roll, and vertical speed and the control board adjusts the motor speeds to stabilize the drone at the required pose. The drone can achieve speeds over $5ms^{-1}$ and its battery provides enough energy up to 13 minutes of continuous flight. The manufacturer provides a software interface, which allows to communicate with the drone via an ad-hoc WiFi network. The driver is compatible with the ROS framework. The API not only allows to set drone required state, but also provides access to preprocessed sensory measurements and images from

onboard cameras. The drone sensory equipment consists of a 6-degree-of-freedom inertial measurement unit, sonar-based altimeter, and two cameras. The first camera with approximately $75° \times 60°$ field of view is aimed forward and provides 640×480 pixel color image. The second one is mounted on the bottom, provides color image with 176×144 pixels and its field of view is approximately $45° \times 35°$. While data from the IDG-400 2-axis gyro and 3-axis accelerometer is fused to provide accurate pitch and roll, the yaw is measured by the XB-3500CV high precision gyro. The pitch and roll precision seems to be better than 0.2° and the observed yaw drift is about 12° per minute when flying and about 4° per minute when in standby (Krajník, Vonásek, Fišer, & Faigl, 2011).

Internal software of the drone not only provides communication, but also takes care of the drone stabilization, and provides so called assisted maneuvers. The bottom camera image is processed to estimate the drone speed relative to the ground, and therefore, the drone is more stable than other quadcopters. After being switched on, an ad-hoc WiFi appears, and an external computer might connect to it using a fetched IP address from the drone DHCP server. Then, the external computer can start to communicate with the drone using the interface provided by the manufacturer. The interface communicates via three channels, each with a different UDP port.

Over the command channel interfaced with ROS, a user controls the drone, i.e., requests it to takeoff and land, change configuration of controllers, calibrate sensors, set PWM on individual motors etc. However, the most used command sets the required pitch, roll, vertical speed, and yaw rate of the internal controller. The channel receives commands at 30 Hz. The navdata channel provides the drone status and preprocessed sensory data. The status indicates, whether the drone is flying, calibrating its sensors, the current type of altitude controller, which algorithms are activated etc. The sensor data contain current yaw, pitch, roll, altitude, battery state and 3D speed estimates. Both status and sensory data are updated at 30 Hz rate. Moreover, the drone can run a simple analysis of the images from the frontal camera and search for a specially designed tags in the images. In the case the tags are detected, the navdata contains estimates of their positions.

In this project, the Parrot AR Drone is configured to track a tag (fig. 2) by using its bottom camera. Details of parameters set to perform tracking of the tag using the in-build tracker of the drone is explained in section 4

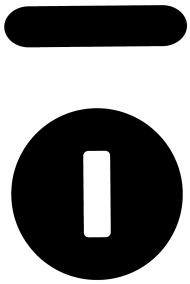

Figure 2: Tag (Roundel)

2.2 Turtlebot

TurtleBot is a low-cost, personal robot kit with open-source software.The main hardware includes:

- Kobuki Base
- Asus Xion Pro Live
- Netbook (ROS Compatible)
- Kinect Mounting Hardware
- TurtleBot Structure
- TurtleBot Module Plate with 1 inch Spacing Hole Pattern

The robotic software development environment of turtlebot includes:

- An SDK for the TurtleBot

- A development environment for the desktop
- Libraries for visualization, planning, and perception, control and error handling.
- Demo applications

The turtlebot can be controlled using the desired control algorithm implemented in ROS framework. In this project the turtlebot is moved in a desired trajectory by publishing the respective velocity command.

3 Control Architecture

The control software architecture of the system implemented in this work is based on the ROS framework as shown in figure 3. The ROS master acts as a

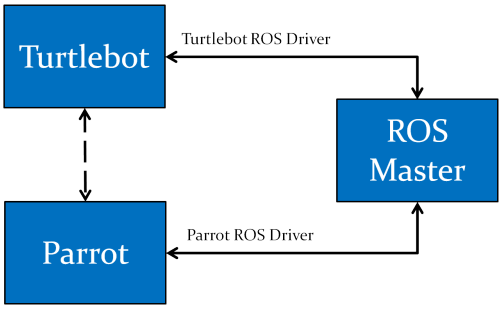

Figure 3: ROS Framework of the System

communication and cooperation media between the Turtlebot and Parrot with the help of their respective ROS compatible drivers.

The detail control software architecture is shown in figure 4. The over all architecture can mainly be divided into four modules:

- Taking off
- Tag Detection and Tracking
- Landing
- General cooperation

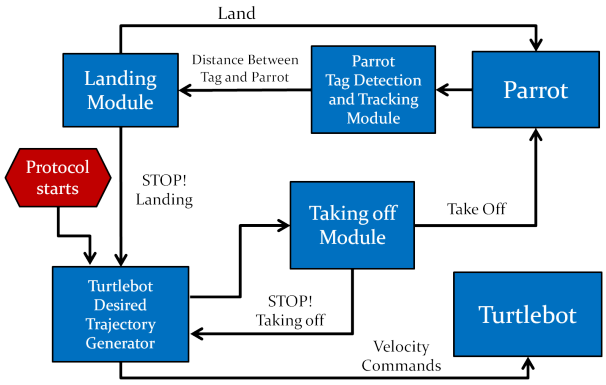

Figure 4: ROS Framework of the System

3.1 Pseudocode Algorithms

1. Turtlebot algorithm

```
< start >
while (not Parrot output: go_home) do
    Turtlebot moves on the path;
    if (Parrot output: stop) then
        while (Parrot not output: move) do
            Turtlebot stops;
            if (Parrot output: go_home) then
                break;
            end if
        end while
    end if
end while
while (not home_position reached) do
    Turtlebot moves to reach home_position;
end while
< end >
```

2. Parrot algorithm

```
< start >
counter1 activation;
...waiting...
counter1 expires;
while (not Turtlebot_stopped) do
    Parrot output: stop;
end while
– Taking off module –
```

```
9: while (not tag_detected and not Parrot_stabilized) do
10:     Parrot takes off;
11:     tag_counter ++;
12:     if (tag_counter >= search_limit) then
13:         Parrot moves to find the tag;
14:     end if
15:     if (tag_counter >= task_limit) then
16:         Parrot lands;
17:         Parrot output: go_home;
18:         exit();
19:     end if
20: end while
21: Parrot output: move;
22: counter2 activation;
23: ...tracking & moving...
24: counter2 expires;
25: – Landing module –
26: while (not tag_detected and not Turtlebot_stopped) do
27:     Parrot output: stop;
28:     if (not tag_detected) then
29:         tag_counter ++;
30:         if (tag_counter >= task_limit) then
31:             Parrot lands;
32:             Parrot output: go_home;
33:             exit();
34:         end if
35:     end if
36: end while
37: while (not Parrot_height == 0) do
38:     Parrot lands;
39: end while
40: Parrot output: go_home;
41: < end >
```

The Parrot AR Drone decides when the Turtlebot must stop and when the task is accomplished sending messages to the Turtlebot. The Turtlebot, which does not use the camera, knows its position based on the odometry (fig. 6). Since we performed all the experiments in a controlled environment, the odometry appears to be pretty robust, but we know that it is not sufficient in real world applications. We focused our project on the cooperation via message exchange and tag recognition. The Turtlebot is just in charge of following a path thought velocity commands (*cmd_vel*) and bring the Parrot AR Drone at the initial position when the flying task is accomplished. In order to reach the initial position, the Turtlebot adopts a kinematic control based on the odometry information.

3.2 Taking Off

As after some experiments, we realized that the Parrot may drift when it takes off, and, even if it is laid on the tag, it is not able to recognize it due to the little shift. An initial protocol forced the drone to stay in the same location, right after taking off, waiting for the tag detection. When the tag is never detected, the Parrot and the Turtlebot are stuck in the same place; just an external operator could solve the problem shooting down the robots or manually moving the Parrot to the right location. So, we decided to improve our drone's algorithm inserting a tag search, right after taking off, as written in the pseudocode (sec. 3.1). The Parrot, if does not meet the tag, starts to span a square, but, if the research takes more than a threshold, the Parrot lands anyway and sends a command to the Turtlebot to go home. Generally the Parrot drifts by some centimetres, so the square spanned by the Parrot should pass over the tag. Otherwise, it means that the Parrot drifted more than what was expected and there is no way to find the tag. This algorithm increase the robustness and the operator safety: even if the operator's computer cannot communicate with the Parrot, it will land by itself without any physical interaction.

3.3 Tag Detection and Tracking

The Parrot AR Drone is configured to detect the tag and track it. As soon as the drone takes off from the Turtlebot, the bottom camera of the drone detects the tag and starts tracking. The in-built tracker of the drone is used in this project because of its robustness. As the resolution of bottom camera of the drone is low, the tracking of tag based on some complex vision based algorithm is difficult. We have tested *visp* (Marchand, Spindler, & Chaumette, 2005) tracking algorithm which was not successful

because of low resolution of the bottom camera.

After the drone is connected with the ROS master, the navigation data of drone can be retrived using ROS topic *"/ardrone/navdata"*.The navigation data (or navdata) is a mean given to a client application to receive periodically ($< 5ms$) information on the drone status (angles, altitude, camera, velocity, tag detection results, etc.). In the stream of NavData, there are informations about vision-detected tags. The goal is to permit to the host to add some functionalities, like augmented reality features. The principle is that the AR.Drone sends informations on recognized pre-defined tags, like type and position. The drone can detect up to four tags or oriented roundel. The kind of detected tag, and which camera to use, can be set by using the configuration parameter *detect_type* (*AR.Drone Developer Guide*, n.d.). The details that can be obtained from NavData are:

- nb_detected: number of detected tags or oriented roundel.
- type[i]: Type of the detected tag or oriented roundel #i
- xc[i], yc[i]: X and Y coordinates of detected tag or oriented roundel #i inside the picture, with (0; 0) being the top-left corner, and (1000; 1000) the right-bottom corner regardless the picture resolution or the source camera.
- width[i], height[i]: Width and height of the detection bounding-box (tag or oriented roundel #i), when applicable.
- dist[i]: Distance from camera to detected tag or oriented roundel #i in centimeters, when applicable.
- orientation_angle[i] : Angle of the oriented roundel #i in degrees in the screen, when applicable.
- rotation[i] : Reserved for future use.
- translation[i] : Reserved for future use.
- camera_source[i] : Camera Source which detected tag or oriented roundel #i.

The drone tries to stay at the middle (500,500) of the detected tag.

3.4 Landing

similar to taking off, landing requires robustness and safety. Whenever the drone wants to land on the mobile platform, it sends a message to the mobile platform to stop. Before starting the landing procedure, the drone checks if it is still detecting the tag and if the Turtlebot already stopped. If the drone does not detect the tag for more time than a threshold, it lands anyway and sends a command to the Turtlebot to go home. On the other hand, if the drone detects the tag and the Turtlebot is not moving, the drone lands on the mobile platform.

3.5 General Cooperation

There is a direct cooperation established via exchange of messages between the UAV and the mobile platform: landing/taking off cooperation. Moreover, there is another indirect cooperation defined by the tag located on the mobile platform, which the drone has to detect and follow.

4 Experimental Results

The experimental setup was carried out in the Robotic Intelligence Lab at Jaume I Unviersity. The tag (roundel), as shown in figure 2, was placed on the top of Turtlebot. The Parrot AR Drone was placed over the tag on the top of Turtlebot. The ROS core running on the computer connected to the Turtlebot was used as a ROS master. The Parrot AR Drone's ROS compatible driver (*ardrone_autonomy*) was run in the ground system which is connected ROS master. The wifi network of AR Drone was used in both robots and the ground system. In this way, we were able to establish a ROS network between the UAV, mobile platform and the ground system through which the cooperation between the drone and moving platform was possible via messages.

The Parrot AR Drone was configured to detect and track the tag on the mobile platform by setting the corresponding parameters (refer to appendix) in the launch file of Parrot AR Drone driver (*ardrone_autonomy*).

Many experiments performing the whole protocol[1,2] were carried, but some additional experiments were conducted in order to test the auto-land mode. These short experiments involve the Parrot finding successfully the tag initially located next to it[3]; the Parrot looking unsuccessfully for the tag until it lands autonomously[4]; the Parrot looking unsuccessfully for the tag, right before landing, until it lands autonomously[5].

Figure 5: UAV and Mobile Platform during the Experiment

The whole protocol consists in placing the mobile platform to its *home* position. Then, the mobile platform was given a linear trajectory with a linear velocity of 0.2 m/s for 5 seconds and then a fixed circular trajectory of radius 0.7 m and a linear velocity of 0.2 m/s. Figure 5 shows the UAV in action when it is tracking the mobile platform. After 1 minutes of tracking, the Parrot AR Drone sends the message to the mobile platform to stop for a while. Then the drone lands on the platform and the mobile platform carries back the drone to its original *home* position using its odometry.

[1] https://youtu.be/7lvlAJ4FWSA
[2] https://youtu.be/5W9rNap42i4
[3] https://youtu.be/TKSGK9J5FgQ
[4] https://youtu.be/TKSGK9J5FgQ
[5] https://youtu.be/NgsPTFTq97A

4.1 General Observations

- The tag detection is activated as soon as the drone takes off
- The drone tries to align itself in the center of the detected tag
- Because of taking the entire part(0-1000) of the detected tag, the oscillation of drone around the center of the detected tag seems high. But, if we consider the precision of tracking, the oscillations can be neglegted.

Figure 6 shows the trajectory followed by the Turtlebot. It starts from the initial position (*home position*) and moves around in a circular path and returns back to the original *home* position with small error (odometry error). This figure shows the accuracy achieved by kinematic control is pretty good.

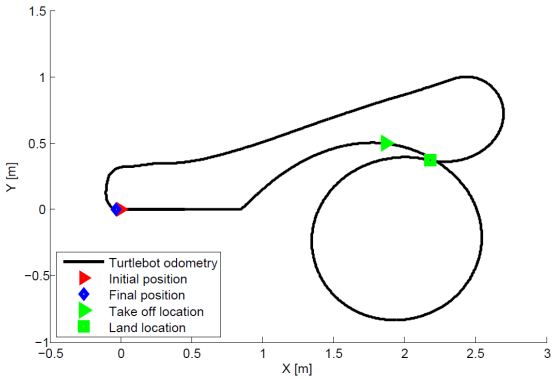

Figure 6: Trajectory followed by the Turtlebot

Figure 7 shows the tracking of the tag using the bottom camera of the drone. The ideal tracking position (500,500) is shown by a black line. The position refers to the point detected by the camera in the tag. The drone's tracking controller tries to align the drone in this ideal tracking position with acceptable

error. The jerks in the plot are due to vibration of the drone while tracking.

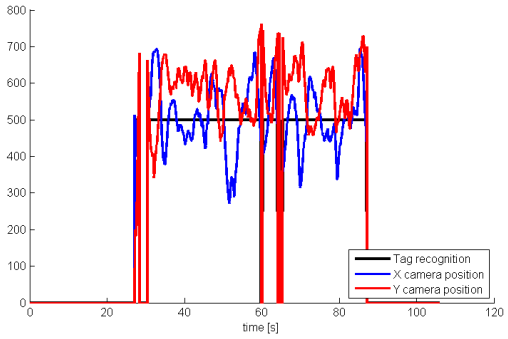

Figure 7: Tag Tracking

Figure 8 shows the tag detection with respect to the motion of the drone and the platform. Whenever the drone takes off, the bottom camera of drone starts detecting the tag and tacking the same. After some time the drone asks the turtlebot to stop and then the drone lands on the turtlebot. Later, the turtlebot carries the drone back to its original *home* position.

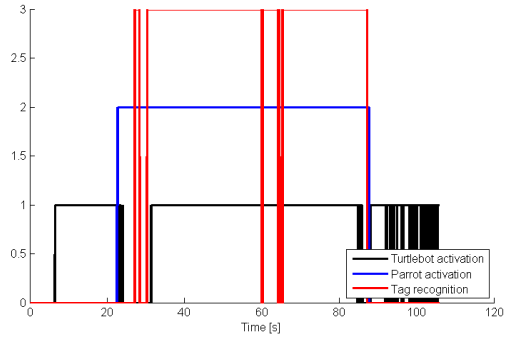

Figure 8: Tag Detection

Figure 9 shows that whenever the tag is not detected, even after the drone takes off, the drones searches for the tag around a square of about 0.7 m with respect to the current hovering position of drone. If it is able to find the tag, it starts tracking the detected tag and sends the command to the mobile platform to move.

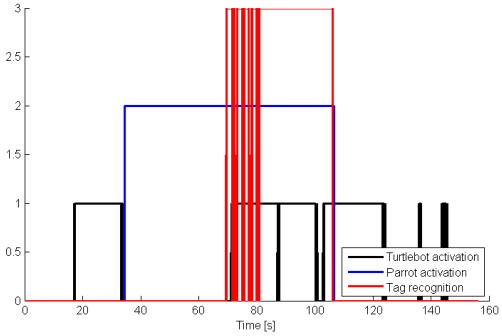

Figure 9: Tag Detection with search for Tag

5 Conclusions

In this work, a modular control software architecture for an UAV and a moving platform has been designed and successfully tested. The cooperation between the drone and the mobile platform has been established to minimize the error during take-off and landing. The on-board tag detection and tracking of the Parrot AR Drone confirms to be robust. All the experiments performed to test the safe autonomous landing proved that our protocol does not need any external supervisor. A detailed documentation[6] on how to setup and run the complete system architecture is also provided.

6 Future Works

In this project, onboard (in-built) tag detection and tracking was used. As an extension of this work, implementation of vision based complete tracking and landing control algorithms by using state estimations and Kalman filter can be done.

[6] https://github.com/emanuelesansebastiano/move_coop_pkg

Appendix

Parameters used for tag detection and tracking:

```
<?xml version="1.0"?>
<!-- Code Tested by Dipendra Subedi and
    Emanuele Sansebastiano, November 2016
     -->
<!-- This is a sample lanuch file, please
     change it based on your needs -->
<launch>
<!-- IPv4 address of your drone -->
<arg name="ip" default="192.168.1.1" />
<param name="flying_mode" value="1" />
<!-- Ultrasound frequency (7 or 8). -->
<arg name="freq" default="8" />
<node name="ardrone_driver" pkg="
    ardrone_autonomy" type="
    ardrone_driver"
output="screen" clear_params="true" args=
    "-ip $(arg ip)">
<param name="outdoor" value="0" />
<param name="max_bitrate" value="4000" />
<param name="bitrate" value="4000" />
<param name="navdata_demo" value="0" />
<param name="flight_without_shell" value=
    "0" />
<param name="altitude_max" value="4000"
    />
<param name="altitude_min" value="50" />
<param name="euler_angle_max" value="0.21
    " />
<param name="control_vz_max" value="700"
    />
<param name="control_yaw" value="1.75" />
<param name="detect_type" value="5" />
<param name="flying_mode" value="1" />
<param name="hovering_range" value="150"
    />
<param name="enemy_colors" value="3" />
<param name="detections_select_h" value="
    32" />
<param name="detections_select_v_hsync"
    value="128" />
<param name="enemy_without_shell" value="
    0" />
<param name="ultrasound_freq" value="$(
    arg freq)" />
<param name="realtime_navdata" value="
    true" />
<param name="realtime_video" value="true"
     />
<!-- Covariance Values (3x3 matrices
    reshaped to 1x9)-->
<rosparam param="cov/imu_la">[0.1, 0.0,
    0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1]</
    rosparam>
<rosparam param="cov/imu_av">[1.0, 0.0,
    0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]</
    rosparam>
<rosparam param="cov/imu_or">[1.0, 0.0,
    0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
    100000.0]</rosparam>
</node>
</launch>
```

Listing 1: Launch File for Ardrone ROS driver

Acknowledgments

The authors thank Prof. Enric Cervera, Dept. of Computer Science and Engineering, for providing valuable advices and support. They also thank the Robotic Intelligence Laboratory at the Jaume I University which houses the Parrot AR Drone, Turtlebot and safe indoor flight arena using which the experiments presented in this work were performed.

References

Ar.drone developer guide. (n.d.). Retrieved from http://web.mit.edu/tinali/www/ARDrone_Developer_Guide.pdf

Bouabdallah, S., Noth, A., & Siegwart, R. (2004). Pid vs lq control techniques applied to an indoor micro quadrotor. In *Intelligent robots and systems, 2004.(iros 2004). proceedings. 2004 ieee/rsj international conference on* (Vol. 3, pp. 2451–2456).

Engel, J., Sturm, J., & Cremers, D. (n.d.). Accurate figure flying with a quadrocopter using onboard visual and inertial sensing. *IMU*, *320*, 240.

Engel, J., Sturm, J., & Cremers, D. (2012). Camera-based navigation of a low-cost quadrocopter. In *2012 ieee/rsj international conference on intelligent robots and systems* (pp. 2815–2821).

Engel, J., Sturm, J., & Cremers, D. (2014). Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems*, *62*(11), 1646–1656.

Gautam, A., Sujit, P., & Saripalli, S. (2015). Application of guidance laws to quadrotor landing. In *Institute of electrical and electronics engineers inc.*

Hoffmann, G. M., Huang, H., Waslander, S. L., & Tomlin, C. J. (2007). Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. of the aiaa guidance, navigation, and control conference* (Vol. 2, p. 4).

Kim, J., Jung, Y., Lee, D., & Shim, D. H. (2014). Outdoor autonomous landing on a moving platform for quadrotors using an omnidirectional camera. In *Unmanned aircraft systems (icuas), 2014 international conference on* (pp. 1243–1252).

Kleusberg, A., & Langley, R. B. (1990). The limitations of gps. *GPS World*, *1*(2).

Krajník, T., Vonásek, V., Fišer, D., & Faigl, J. (2011). Ar-drone as a platform for robotic research and education. In *International conference on research and education in robotics* (pp. 172–186).

Majumder, S., & Shankar, R. (2014). Moving object tracking from moving platform. In *Signal processing and integrated networks (spin), 2014 international conference on* (pp. 85–89).

Marchand, É., Spindler, F., & Chaumette, F. (2005). Visp for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics & Automation Magazine*, *12*(4), 40–52.

Prevost, C. G., Desbiens, A., & Gagnon, E. (2007). Extended kalman filter for state estimation and trajectory prediction of a moving object detected by an unmanned aerial vehicle. In *2007 american control conference* (pp. 1805–1810).

Sarris, Z., & Atlas, S. (2001). Survey of uav applications in civil markets. In *Ieee mediterranean conference on control and automation* (p. 11).

Wenzel, K. E., Masselli, A., & Zell, A. (2011). Automatic take off, tracking and landing of a miniature uav on a moving carrier vehicle. *Journal of intelligent & robotic systems*, *61*(1-4), 221–238.