Quadrotor pilot training using augmented reality

Semester project

Krzysztof Lis

liskr@ethz.ch

Advanced Interactive Technologies Lab
ETH Zürich

Supervisors:
Dr. Fabrizio Pece, Christoph Gebhardt
Prof. Dr. Otmar Hilliges
Prof. Dr. Roger Wattenhofer

December 9, 2016

Acknowledgements

I would like to sincerely thank those whose contributions helped me complete this project:

- Christoph, Isa, Jakob H, Kamila Součková, Michelle, Richard Královič for participation in the user study and feedback.
- Epic Games and other Unreal Engine developers, for providing the Unreal Engine 4 framework [1] and associated graphical assets.

Abstract

With the aim of improving the experience and effectiveness of human micro aerial vehicle pilots, especially in confined spaces and in presence of obstacles, we have developed a training application involving maneuvering a real quadrotor around virtual obstacles in an augmented reality setting as well as a piloting interface using a virtual reality headset and motion controllers to provide and intuitive control method.

Our user study shows that the motion controller interface offers similar user experience and performance to a traditional gamepad control scheme despite requiring just one hand to operate. The augmented reality application was perceived as attractive and novel by the users.

Contents

Chapter 1

Introduction

Micro aerial vehicles (MAV) have found a variety of uses such as photography, remote exploration, entertainment. However, their weakness is vulnerability to collisions, which can break the rotors or destabilize the vehicle and cause it to fall and suffer heavy damage.

This limits the practicality of drones in confined spaces such as buildings. However, there are cases where drones would be useful in indoor environments: exploration, inspection, mapping, search and rescue operations. Therefore a solution improving their indoor performance would be valuable.

There is ongoing research in a variety of automatic obstacle detection and avoidance systems. However in some cases a human pilot is still needed: existing drones not equipped with necessary sensors or software for autonomous flight, unpredictable tasks requiring flexibility of command, or when a user pilots the vehicle for entertainment.

In this work we explored ways to improve the experience and effectiveness of human drone pilots. With the aim to provide a safe way of training piloting skills, we developed a game involving navigating a real drone in an augmented reality environment. In addition to training, we attempted to make drone control more intuitive by creating a piloting mechanism based on hand gestures measured by motion controllers.

Chapter 2

Related work

2.1 Training through simulation

Simulation is a natural approach to training and MAV simulation software has been developed: [2], [3]. However, a remotely controlled quadcopter is a complex system both from physical and software perspectives. It is challenging to faithfully predict and simulate undesirable circumstance like drone drifting or delays in communication - and such unexpected events can be the cause of a collision. When flying a real drone the pilot can experience the flaws of the specific quadrotor model and learn how to handle them.

2.2 Display of drone's video in head mounted displays

Existing products: [4], [5] prove that head mounted displays are a valid approach to presenting the video feed captured by the drone. There is also a drone model which can be used to play augmented reality games [6]. These solutions often use mobile devices as screens and analog sticks for control. By utilizing high quality virtual reality equipment with motion tracking we try to explore if these devices can improve control of drones and the presentation of the video stream.

2.3 Gesture control

Gestures (measured using body pose tracking) can be used instead of traditional input devices to intuitively control a quadcopter [7]. We attempt to improve the accuracy of control by using precisely tracked motion controllers instead of body poses.

Chapter 3

Design

We consider a scenario where the pilot controls the drone remotely and observes the video transferred in real time from its camera (either on a screen or head-mounted-display). In order to help pilots fly their drones effectively, precisely and safely, we have designed the following solutions:

- Training application utilizing augmented reality to allow users to safely train obstacle avoidance - described in Section 3.1.
- An intuitive method for controlling drone movement using motion controllers and virtual reality equipment - described in Section 3.2.

3.1 Training application design

The application is designed to train the user's ability to effectively navigate a quadcopter in a confined space in presence of obstacles. To achieve a realistic experience but prevent the danger of collision, we use augmented reality: the user controls a real quadcopter flying in an empty room, and virtual obstacles and goals are added to the video feed.

The pilot's task in the game is flying through a path marked by virtual rings and avoid collisions. An example view of the game can be seen in Figure 3.1. The game records the following metrics which the user can strive to improve:

- c - number of collisions with obstacles - they impact the score negatively,
- g - number of rings the drone has successfully flown through - they impact the score positively,
- $s = g - c$, score,
- t - time spent to complete the task - measured from quadcopter takeoff to landing.

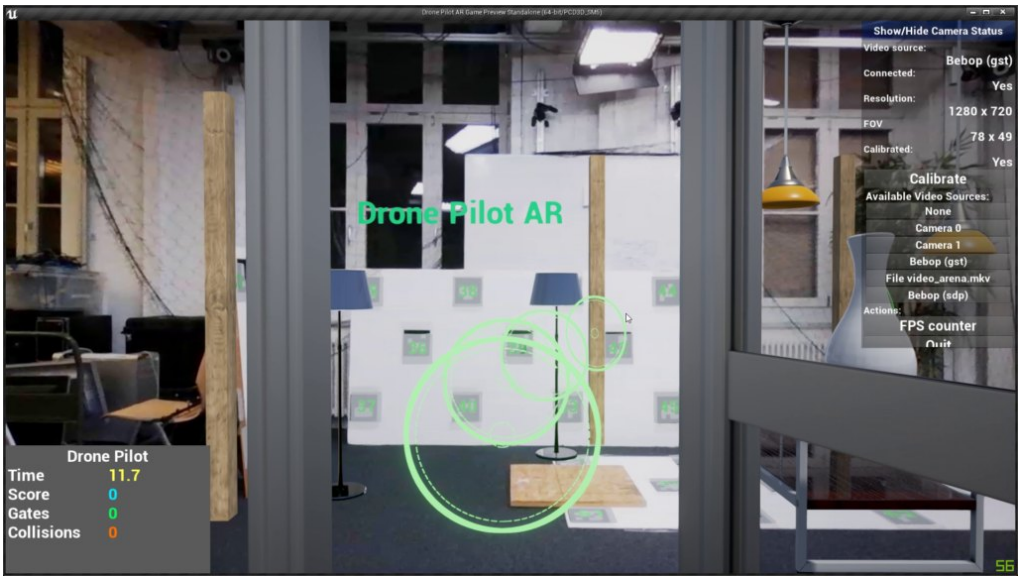


Figure 3.1: View of the augmented reality training game. The pilot sees the world from the quadcopter's perspective and virtual objects are added to the video stream. The task is to fly through the green rings while avoiding collisions with obstacles (doors, furniture). In the bottom left the counters for time, score, number of collisions, number of passed rings are visible.

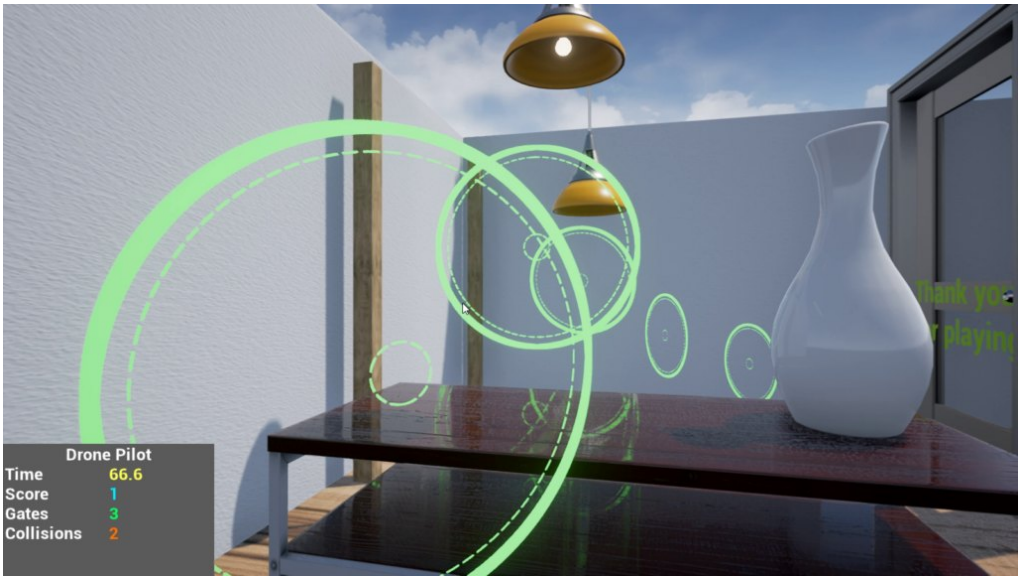


Figure 3.2: View of the training game in simulation mode, which can be used to get accustomed with the controls or practice without access to a real quadcopter.

Figure 3.3: Layout of buttons on the motion controller used in our application. The activation pad is a touch sensitive sensor which can detect whether the user is touching it - without need for a strong press. Movement commands will only be sent while it is being touched.

Successfully passing a ring is indicated by a sound signal and collision is communicated through vibration of the gamepad or motion controller.

The application also provides a simulation mode which can be used to get accustomed with the controls or practice without access to a real quadcopter, but no emphasis is placed on accurately simulating the drone's physics. A view of this mode can be seen in Figure 3.2. The drone can be controlled with analog sticks (example control scheme shown in Figure 5.1) or with motion controllers using the method described in Section 3.2.

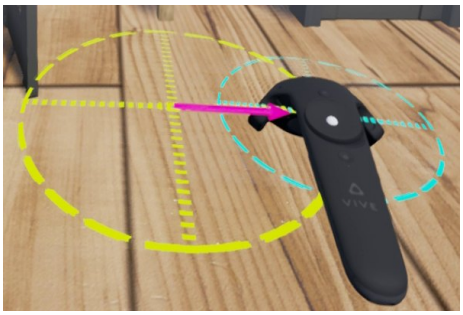
3.2 Intuitive drone control with motion controllers

Traditional control schemes such as analog sticks require the user to translate the desired drone movement into a different move of fingers. In order to make the the process more intuitive and accessible to inexperienced users we propose a method where pilot's hand moves are directly mapped to drone movement:

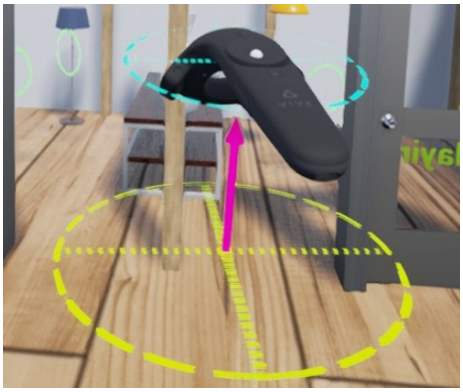
- The user holds a motion controller whose pose is precisely tracked
- The controller's buttons are used to take off, land and start or stop movement. An example layout of buttons is shown in Figure 3.3.
- The drone does not move when the user is not touching the activation pad. The pilot can quickly stop the drone by releasing this pad.
- When the activation pad is touched, the controller's current pose is recorded as *origin position* and *origin rotation*. When the user moves the controller

(a) Inactive state - activation pad is not touched, no movement commands are issued.

(b) Activation - the pad has been just pressed, *origin pose* is stored and represented with the green marker.



(c) Command to move right.



(d) Command to ascend.

(e) Command to turn the drone to the left. Angular speed is proportional to the angle between controller's axis and forward vector of origin pose.

Figure 3.4: Principle of the intuitive control mechanism. Initially the control is inactive 3.4a. To issue a movement command, the user touches the activation pad 3.4b and displaces the controller. Linear movement speed is proportional to the distance to origin pose (pose at time of activation): 3.4c, 3.4d. To turn the drone, the user rotates controller around the vertical axis 3.4e.

from that position, the drone moves in the direction determined by the vector from origin position to current position and speed proportional to distance between those points. When the user turns the controller right or left around the vertical axis, the drone rotates with angular speed proportional to the displacement angle. Examples of the principle are shown in Figure 3.4.

A quadcopter's movement has 4 degrees of freedom: 3 dimensions of movement in space and 1 dimension of rotation around vertical axis as it must be stabilized along the remaining axes. An analog stick provides 2 degrees of freedom, so two sticks and thus both hands are needed to control a quadcopter. A single motion controller has 6 degrees of freedom: 3 spatial and 3 rotational - and therefore only one is needed to pilot the drone. The other hand and controller is left free to perform a different task.

Chapter 4

Implementation

We implemented the training application using *Unreal Engine* [1] game engine as it provides high quality realistic rendering, built-in integration with a variety of virtual reality equipment, and collision detection. It does not have an augmented reality feature, so we have created a plugin for that purpose - described in Section 4.1.

We have chosen the *HTC Vive* virtual reality equipment which includes precise motion controllers and is known to integrate with Unreal Engine applications. We have also attempted to evaluate an Oculus Rift DK2 headset, however the driver provided by the manufacturer failed to connect to the headset on our development machine.

We used the *Parrot Bebop 2* quadcopter because it supports high resolution video streaming and has a simple control API through the producer-provided SDK. Section 4.2 describes the architecture of communication between the drone and training application.

4.1 Augmented reality plugin for Unreal Engine

We have developed a plugin for Unreal Engine enabling development of augmented reality applications. Since that functionality may be useful for other developers, we have released the plugin publicly under an open source license [8]. The plugin provides the following features to an Unreal Engine application:

- acquisition and presentation of video,
- camera pose tracking using fiducial markers,
- camera calibration,
- visual editor for fiducial marker configurations.

Our implementation relies on the OpenCV[9] library.

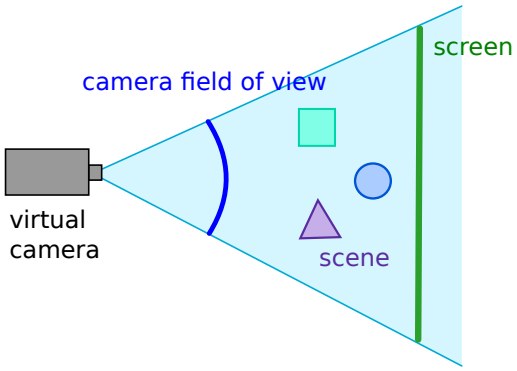


Figure 4.1: Arrangement of objects rendered by the graphics engine to create an augmented reality experience: The video from camera is displayed on the screen plane, the screen is kept positioned in front of the camera but behind the scene objects. The size of the screen is adjusted to fill the whole field of view of virtual camera.

4.1.1 Acquisition and presentation of video

Video stream is acquired using OpenCV's `VideoCapture` class, which is capable of receiving video from cameras, files and network streams.

The drone will be flying in an empty room, so all virtual objects are in front of real world background and the video should be displayed behind the virtual scene (even if that was not the case, simulating occlusions by real world objects is a difficult problem and beyond the scope of this project). This is achieved by displaying the video on a plane in the virtual world, placed behind the scene. The screen is moved to be always in front of the virtual camera. The arrangement is shown in Figure 4.1.

To ensure proper alignment of the video and virtual scene, the field of view angle of the camera used by the graphics engine must match the field of view angle of the real camera. To that end, we perform calibration of the camera using OpenCV's calibration feature with the asymmetric circle grid pattern.

4.1.2 Tracking

To track the camera pose, we use fiducial markers and the implementation of ArUco[10] method provided by OpenCV[11]. Given the definition of the spatial configuration of markers and an image of a scene containing this marker arrangement, the algorithm calculates the camera pose.

Coordinate frame conversion

However, OpenCV uses a right-handed coordinate system while Unreal Engine operates on a left-handed coordinate system. To convert between them, we swap the X and Y coordinates and negate rotation angles.

The ArUco algorithm calculates the transformation from the marker board reference frame to the camera frame[11, Pose Estimation], let us represent it with translation vector t and rotation matrix R. For a given point in space, we can write the transformation between the point's coordinates p_m in marker board frame and its coordinates p_c in the camera frame:

$$p_c = Rp_m + t. \tag{4.1}$$

For augmented reality, we want to know the camera pose in the virtual world. We assume that the marker board is placed at the origin of the virtual scene, so the marker's frame is equivalent to the virtual world coordinate frame. The camera is located at the origin of camera's frame - we set $p_c = 0$ in equation 4.1:

$$0 = Rp_m + t$$

and by solving for p_m we obtain:

$$p_m = -R^{-1}t$$

which is the camera location in the virtual world. The camera's rotation in the virtual scene's frame is equal to $R^{-}1$.

Outlier detection

The pose estimation is burdened with noise, so we perform smoothing to calculate the camera pose $C(t)$ at frame t:

$$C(t) = \alpha m(t) + (1 - \alpha)C(t - 1), \tag{4.2}$$

where $m(t)$ - pose measured at frame t by the tracker, $C(t-1)$ - camera pose at previous frame, α - smoothing factor, in our implementation $\alpha = 0.1$.

However, sometimes the algorithm returns obviously anomalous values, we consider a pose to be an outlier and discard it when it satisfies one of the conditions:

- the roll angle (rotation around the drone's forward axis) exeeds threshold r_max. The drone we used in this experiment automatically stabilizes its camera such that changes in pitch and roll are not seen in the stream. Therefore if the tracker detects a high roll angle, the result must be incorrect.

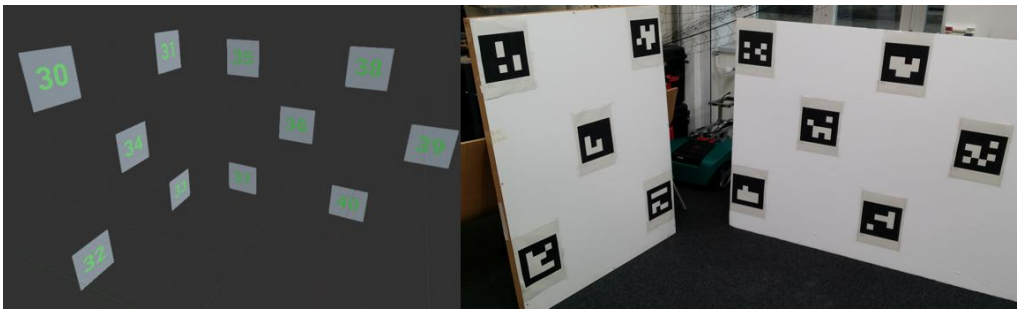


Figure 4.2: A fiducial marker configuration in the editor (left) and in reality (right). The editor simplifies creation of boards containing many markers with varied spatial poses. Each marker is identified by its unique number, which is shown in editor as well as in the image for printing.

- If the distance between currently measured pose and pose detected in previous frame exceeds d_max - the distance between consecutive poses is limited by drone's velocity.

In our implementation we used $r_max = 10^0$ and $d_max = 30$cm.

4.1.3 Visual editor for fiducial marker configurations

To create enough space for a quadcopter to fly, we had to place a big number of fiducial markers. The tracking process requires knowledge of positions of all markers. We implemented a mechanism to create and edit a spatial configuration of markers using Unreal Engine's editor, which provides convenient controls for moving 3D objects. The example marker configuration is shown in Figure 4.2.

Our game then saves files containing the marker images and the user can print them and arrange them in the real space. If markers need to be moved or added later, it can be conveniently done with the editor.

4.2 Communication and control of Bebop 2 drone

The implementation challenge in this project comes from the need to integrate a wide range of software and hardware. The system architecture is shown in Figure 4.3 and consists of the following components:

- *Bebop 2* quadcopter - communicates with the controlling computer using WiFi, receives piloting commands: desired linear and angular velocity and signals to start or land, while transmitting a video stream and status information, including position and velocity obtained through odometry.

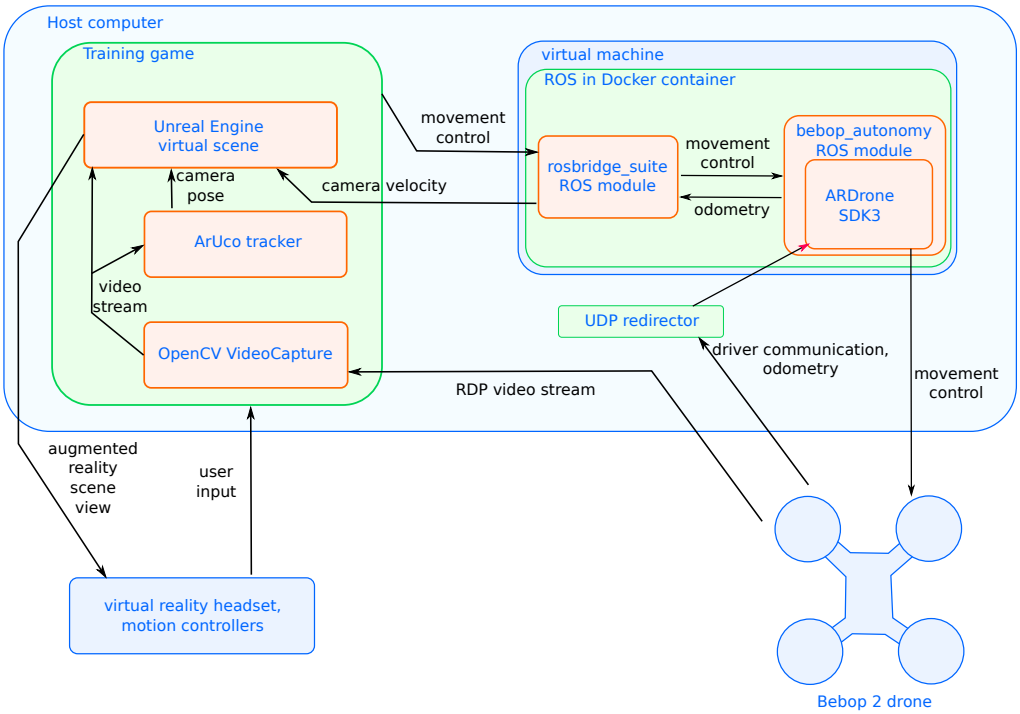


Figure 4.3: The components of the augmented reality training application. Black arrows represent information flow.

- *Drone driver* - software running on the computer dedicated to interfacing with the drone, at its core it contains the *ARDrone SDK 3* which interprets the drone's messages and sends commands to it. Additionally, we use the Robot Operating System (ROS) along with `bebop_autonomy`[12] module which provides a ROS interface for Bebop SDK. This should allow using extending the project to use a different drone in the future, provided it can be controlled using a ROS module, which is likely due to the popularity of ROS in robotics research. Communication with the training application is performed by exchanging JSON messages over a TCP connection using the `rosbridge_suite`[13] module.

  Drivers for the virtual reality equipment are so far only available for Windows operating system which forces us to use it on the host computer. However, the drone SDK and ROS require a Linux environment, so we execute them inside a virtual machine as a Docker container. This architecture allows us to develop and test the software on Linux and then transfer the image and easily deploy it inside the virtual machine.

- *UDP Redirector* - the drone sends messages to the IP address of the computer connected through WiFi, but the drone SDK runs inside a virtual machine which has a different IP, so we implemented as simple program

that forwards the packets to the virtual machine. In contrast, the video stream should received by the training application - video packets are separated from control packets by their destination port number.

- *Training application* - the game described in Section 3.1. It forwards the user's commands from input devices to the drone driver and displays the video stream transmitted by the drone.

4.2.1 Utilizing drone's odometry

The drone provides an estimation of its linear and angular velocity which is more reliable in the short term than the fiducial marker tracker. We take advantage of it and perform a simple sensor fusion: the estimated drone pose $D(t)$ at time t is a weighted average of the most recent pose $C(t)$ reported by marker tracker (defined in equation 4.2) and an estimation based on drone's odometry:

$$D(t) = \beta C(t) + (1 - \beta)(D(t - \Delta t) + v(t)\Delta t),$$

where: Δt - time between rendering of consecutive frames in the engine, $D(t - \Delta t)$ - estimated pose at previous frame, $v(t)$ - current drone velocity estimation provided by the driver, β - configurable constant.

Chapter 5

Evaluation

We have evaluated the performance of our solutions in a user study.

- Which drone control method, analog stick or motion controller, is easer to operate?
- Is our augmented reality application an engaging way of training drone pilots?

5.1 User study design

We compared the motion controller based piloting scheme described in Section 3.2 to a usual gamepad device with two analog sticks with the control mappings shown in Figure 5.1. The control mechanisms' ease of operation was measured with a NASA "raw" Task Load Index [14] survey, which assesses the task's difficulty in six categories: mental demand, physical demand, temporal demand, performance achieved, effort required and frustration.

Their effectiveness was compared using the completion times and scores achieved by participants in the game, the scoring system is described in Section 3.1.

The augmented reality training application's appeal was evaluated with the User Experience Questionnaire [15].

First, the training application in simulation mode is used to compare the control methods:

- Determine the order of control methods for user. To avoid bias, odd numbered participants begin with motion controller method while the remaining participants begin with analog sticks.
- Training: the user plays the game in simulation mode to become familiar with the control methods in the previously determined order, no measurements are performed.

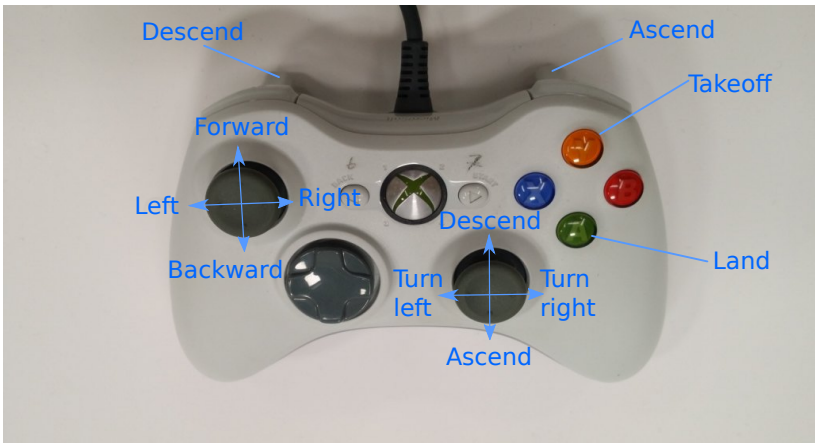


Figure 5.1: The traditional analog stick control scheme utilized in the user study.

- First test run: the user is asked to complete the game using the first control method, time and score is measured.
- First survey: the user is asked to evaluate their general skill with the first control method (choices: *no experience*, *I use it sometimes*, *I use it often*) and answer the Task Load Index questions about this control method.
- Second test run and survey: similar to previous two steps but using the second control method.

Then the participant uses the augmented reality application with a real drone, with the control method of their choice. Time and score is not recorded because it may be altered by random events unrelated to the pilot's actions such as failure of pose tracking. After playing the augmented reality game, the user evaluates it by answering the User Experience Questionnaire.

5.2 Results

The user study had 6 participants. According to their answers to the question about skill - shown in Figure 5.2 - the majority had none to moderate experience with the control methods.

5.2.1 Control method comparison

The results of the raw Task Load Index survey are displayed in Figure 5.3. The motion controller was on average slightly less mentally demanding and users felt their results were better, but they also reported being under more time pressure when using that interface. The mean scores and times achieved by the players are

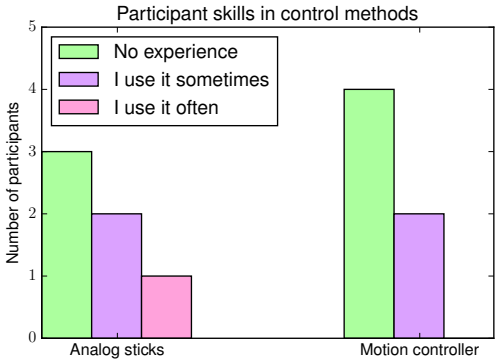


Figure 5.2: The participants' reported levels of proficiency in the control interfaces used in the experiment.

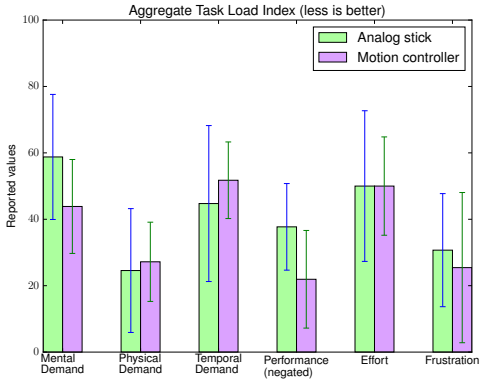


Figure 5.3: Mean Task Load Index scores in different categories for the two control methods. Lower values mean a more positive evaluation. The error bars represent one standard deviation from the mean.

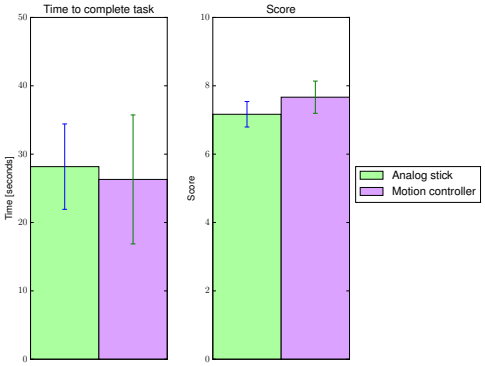


Figure 5.4: Mean values of time spent to complete the task and scores achieved by participants while playing the training game using different control methods. Score calculation is described in Section 3.1, higher score and lower time is better. The error bars represent one standard deviation from the mean.

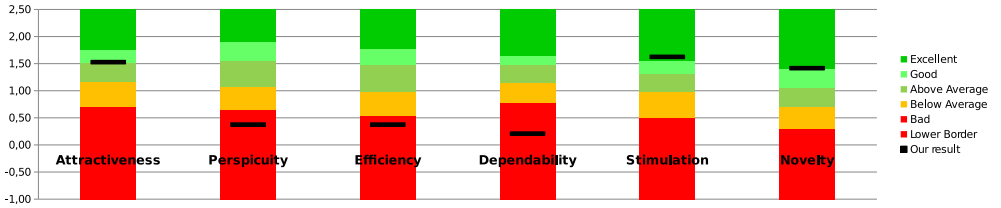


Figure 5.5: Scores describing the augmented reality application obtained through the User Experience Questionnaire, compared to benchmark data from other studies using this questionnaire. Stimulation and novelty are in the range of 10% best results and attractiveness is in top 25% best results while efficiency, dependability and perspicuity (clarity) are in range of 25% worst results.

shown in Figure 5.4. Overall the performance of both interfaces was comparable, despite the motion controller requiring just one hand to operate.

5.2.2 Augmented reality application evaluation

We used the answers to the User Experience Questionnaire to calculate scores of the augmented reality application in six categories: attractiveness, perspicuity (clarity), efficiency, dependability, stimulation, novelty. The scores are shown in Figure 5.5 in comparison to existing benchmark data from other studies using this questionnaire.

The results clearly show that that users find the application novel, exciting and attractive, but it is not efficient and can not depended on to perform its task well. We believe the negative results in those categories are caused by the following flaws in the application:

- long delay on video transmission and processing means the pilot needs to wait to see the effect of their inputs, and can not control the drone smoothly,
- the fiducial marker tracker sometimes fails to report a correct pose for several seconds, causing the virtual objects to be wrongly aligned with the real world.

We acknowledge the existence of these problems and suggest ways to improve the solution in Section 6.1.

Chapter 6

Results and discussion

6.1 Future work

The evaluation revealed several flaws in the solution and we propose ways in which they could be addressed.

Fiducial markers have proven to be impractical at the scale of a whole room, as it was required to place many markers to ensure enough of them were always visible from the drone's camera. The markers are also not recognized if the distance to them is too high. Their positions must be measured and provided to the tracker and at this scale the precision of measurements was limited. Despite our efforts, the tracker frequently reported wrong poses, which happens rarely for smaller scales and numbers of markers. Therefore we suggest implementing tracking drone position with other means, such as keypoint-based visual odometry or a VICON system if available.

Furthermore, the drone is stabilized and the range of possible poses is limited. This allowed us to detect outlier poses, but it would be more efficient to use it as a constraint for the equation that calculates poses from point correspondences. Drone pose tracking could be also made more reliable against outliers and noise if a more sophisticated sensor fusion method was used to combine information from fiducial markers and drone's odometry. The Extended Kalman Filter is worth exploring, but requires an estimate of measurement uncertainty.

The video transmission and processing pipeline should be examined with aim to reduce the input delay. We suspect unnecessary buffering is happening at some point of the process.

Finally, alternative ways to use the motion controllers for drone piloting could be researched. We did not use the controller's ability to measure tilt around forward and right-left axes but some users suggested this would be a natural metaphor for the tilting movement which the drone uses to move horizontally.

6.2 Conclusions

We explored ways to improve the experience and effectiveness of human drone pilots, especially in confined spaces and in presence of obstacles. We have implemented a piloting interface using a virtual reality headset and motion controllers and a training application about maneuvering a real drone around virtual obstacles in an augmented reality setting. We release a part of this software as a plugin which might be useful for other developers.

We have tested the solutions in a user study. We observed comparable user experience and performance of the motion controller interface and traditional gamepad controls. The motion controller is operated with only one hand while both hands are needed to use a gamepad.

The users perceive the augmented reality application as unreliable and inefficient, presumably due to delay in video transmission or processing and faults of the pose tracking system. On the other hand, they find the solution attractive and novel, which means augmented reality is an engaging method of interacting with drones and the direction should be pursued further.

Bibliography

[1] Unreal Engine 4. https://www.unrealengine.com

[2] Furrer, F., Burri, M., Achtelik, M., Siegwart, R.: RotorS—A Modular Gazebo MAV Simulator Framework. In: Robot Operating System (ROS): The Complete Reference (Volume 1). Springer International Publishing, Cham (2016) 595–625

[3] Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., von Stryk, O.: Comprehensive simulation of quadrotor uavs using ros and gazebo. In: 3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR). (2012) to appear

[4] Parrot SA: Parrot DISCO FPV. https://www.parrot.com/us/drones/parrot-disco-fpv

[5] Aras, K.: CloudlightFPV for Parrot Bebop. http://cloudlightfpv.com/bebop.php

[6] Walkera Technology Co., Ltd.: AiBao - game drone. http://www.walkera.com/index.php/Goods/info/id/42.html

[7] Pfeil, K., Koh, S.L., LaViola, J.: Exploring 3d gesture metaphors for interaction with unmanned aerial vehicles. In: Proceedings of the 2013 International Conference on Intelligent User Interfaces. IUI '13, New York, NY, USA, ACM (2013) 257–266

[8] Lis, K.: Augmented Unreality - augmented reality plugin for Unreal Engine 4. https://github.com/adynathos/AugmentedUnreality

[9] Itseez: OpenCV - Open Source Computer Vision Library. https://github.com/itseez/opencv (version 3.1.0).

[10] Garrido-Jurado, S., noz Salinas, R.M., Madrid-Cuevas, F., Marín-Jiménez, M.: Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition **47**(6) (2014) 2280 – 2292

[11] OpenCV - Detection of ArUco Markers. http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html

[12] Monajjemi, M.: Bebop Autonomy - ROS driver for Parrot Bebop drone, based on Parrot's official ARDroneSDK3. http://wiki.ros.org/bebop_autonomy

[13] Mace, J.: Rosbridge Suite - a JSON interface to ROS. http://wiki.ros.org/rosbridge_suite

[14] Hart, S.G., Staveland, L.E.: Development of nasa-tlx (task load index): Results of empirical and theoretical research. In Hancock, P.A., Meshkati, N., eds.: Human Mental Workload. Volume 52 of Advances in Psychology. North-Holland (1988) 139 – 183

[15] Laugwitz, B., Held, T., Schrepp, M. In: Construction and Evaluation of a User Experience Questionnaire. Springer Berlin Heidelberg, Berlin, Heidelberg (2008) 63–76