
Kyoto University
Graduate School of Engineering
Mechanical Engineering Science

# A UAV Teleoperation System Using Subimposed Past Image Records

A thesis presented for the degree of
Master of Science

Jean Nassar
*Author*

Dr. Fumitoshi Matsuno
*Supervisor*

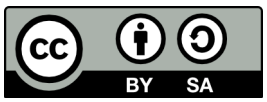
March 31, 2017

*A UAV Teleoperation System Using Subimposed Past Image Records*
*過去画像を用いた飛行ロボットの遠隔操作手法*

All the source code for this project is accessible at the author's public Github profile at `https://github.com/masasin/spirit`.

This document was typeset using the XeTeX typesetting system created by the Non-Roman Script Initiative and the memoir class created by Peter Wilson. The text is set in 10pt in Linux Libertine O. The other fonts used include Linux Biolinum O, `DejaVu Sans Code`, and IPA Mincho (明朝).

The PDF version of this thesis is fully hyperlinked and navigable.



because [...] if you don't know where you are, then you don't know where you're going. And if you don't know where you're going, you're probably going wrong.

---

*I Shall Wear Midnight*
Terry Pratchett

**Summary**

Subimposed Past Image Records Implemented for Teleoperation (SPIRIT) is a novel robotics teleoperation system which overlays the current position and orientation of a vehicle onto previously acquired images. This research focuses on developing a SPIRIT-based user interface for aerial robots. The proposed method combines field of view (FOV) information with state estimation, and selects a suitable image to use as a background. It works even in a low-throughput environment where video data might suffer, and only requires one camera.

Experiments were conducted to study the efficacy at increasing accuracy while decreasing time and wasted movements. Participants were tasked with flying a drone to a location above a known target while using the proposed system, and comparing the results with those obtained by flying with a slowed-down onboard camera.

SPIRIT significantly improved accuracy from 0.667 m to 0.403 m, an improvement of almost 40%, and reduced the overall workload by 37.5%, especially in the physical, temporal, and performance metrics. It also significantly improved the users' control and awareness of absolute and relative positioning. Their self-assessed ability to understand their position relative to the target increased by 86.9% from an average rating of 2.556/7 to 4.778/7.

There were non-significant increases in time and path lengths, but improvements over subsequent runs indicate that these are simply an issue of familiarity with the system. Further testing is needed to verify the effect of SPIRIT on completion time and path length, and its efficacy in various environments.

Overall, the tests were successful, and all participants gave praise for the system, as well as valuable feedback.

**Acknowledgements**

I would first like to thank my thesis advisor, Professor Fumitoshi Matsuno for his guidance during my past several years here. He never gave up on me even when I was discouraged. I would also like to thank the staff and students at Kyoto University for their valuable advice and companionship. From building robots to intramurals, from dinner invitations to hugs, hiking and flying, I had an unforgettable experience. Thank you especially to the participants in my research, who volunteered their precious time and gave good feedback.

Furthermore, I would like to thank my friends who stood by me, no matter where they are in the world. I am particularly indebted to Chihiro for her great assistance and selfless sacrifice. In addition, I am grateful to the Makimuras, as well as the other members of KAHF, who became like a second family to me, a home away from home.

I also thank the Kiyomizudera Temple for awarding me a scholarship, which helped reduce the burden.

Finally, I would like to express my sincere gratitude to my loved ones, who were with me every step of the way. This accomplishment would not have been possible without them.

There are many more people who deserve thanks. So to everyone who helped support me, knowingly and unknowingly, directly and indirectly, you have my appreciation.

# Contents

# List of Figures

# List of Tables

# Part I

# Overview

# 1 Introduction

This report is divided into six parts. In Part I, the field is introduced and the purpose of the research is stated. The background of the problem and previous research are also presented. In Part II, the proposal is shown and the new method's implementation is explained in detail. In Part III, the experiments are described and the results presented. The conclusions are discussed in Part IV. Appendices, including the system details, are in Part V. Part VI contains the glosaries and references.

## 1.1 Unmanned Aerial Vehicles

### History

Unmanned aerial vehicles (UAVs), also known as drones if they have artificial intelligence (AI), have been used since at least the mid-19th century, when the Austrian military bombarded the Italian city of Venice.[1] Later developments included aerial torpedoes, assualt drones, and target drones, controlled both autonomously or remotely. Reconnaissance drones were developed in the 1960s, and used heavily in Vietnam. The importance of battlefield UAVs was realized in 1982 with their successful utilization by the Israeli Air Force against the Syrian Air Force over Lebanon.[2]

Modern drones have both military and civilian uses. Multirotor UAVs have recently become common with the public, and are now used recreationally and professionally. Some of the applications of UAVs include:

- capturing photos and videos
- racing
- site inspection
- agriculture and land assessment
- news gathering
- police surveillance
- exploration
- search and rescue, and disaster relief

### Issues with modern drones

A modern UAV's payload typically consists of at least one camera, an inertial measurement unit (IMU) with at least six degrees of freedom (DOFs), and an autopilot to aid with stability. More sophisiticated UAVs might have additional cameras, or sensors for altitude and distance, including those for 3D reconstruction of the environment. However, these features tend to require a larger, heavier UAV, which might cost more and be less maneuverable.

Teleoperated UAVs usually use either line of sight, where the pilot has to be able to see the drone directly, or first person view (FPV), where the pilot uses a video downlink from the drone to control it. The downlink can be displayed in video goggles or on a standard display. Some setups allow the camera on the drone to gimbal, while others have a zoomed-in fisheye lens, either of which can provide an immersive experience when combined with head tracking. Figure 1.1 shows various examples.

(a) A transmitter.[3] (b) A phone.[4] (c) FPV goggles.[5]

Figure 1.1: Various methods of displaying a FPV feed.

FPV, however, is normally 2D, and does not afford the user any depth perception. As a result, loss of situational (i.e. spatial) awareness can occur, and the boundaries of the vehicle with respect to the environment might not be known. This is known to have contributed to multiple crashes into the environment in the past, with powerlines damaged,[7] and people gravely injured.[8, 9] Incidents of UAVs flying dangerously close to airports have also been reported.[10]

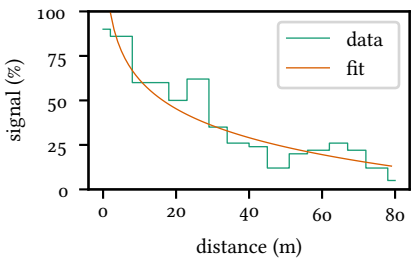



Figure 1.2: Signal strength of the AR.Drone with respect to distance. Extracted from [6].

Moreover, degradation of signal (DOS) can occur when flying far, or in an environment with bad reception, causing dropped frames and choppy video. The operator may then miss important cues and react too late, leading to an accident. Figure 1.2 shows the severe DOS that occurs with the drone used in this research, even at intermediate distances. By 50 m, complete loss of control can occur.

## 1.2 Research objective

The objective of this research is to increase the situational awareness of a UAV pilot using a single onboard monocular camera. The system should be capable of functioning in a low bandwidth situation, and provide increased positional control compared to FPV.

Other objectives would be faster time to completion of the task, more direct paths, and a lower task load for the operator.

# 2 Background

## 2.1 Prior work

Several methods to show a robotic avatar in a virtual environment were proposed.

Nielsen, Goodrich, and Ricks [12] presented an interface in which a 3D model of the robot, a video stream from a mounted camera, a map, and pose information, were combined in a mixed-reality display. Kelly et al. [13] extended the concept to an outdoor environment, and succeeded in making their system work in various low-throughput situations. However, most methods require many sensors, such as laser rangefinders (LRFs), which add expense and weight to the vehicle.

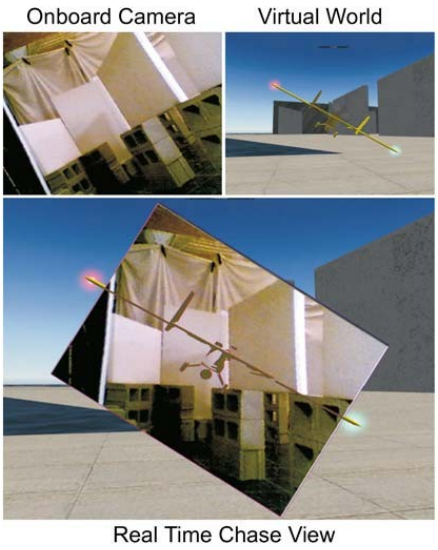



Figure 2.1: Hing et al.'s chase view.[11]

Hing, Sevcik, and Oh [11] succeeded in making a chase view for a fixed-wing UAV, as shown in Figure 2.1. This showed the orientation of the drone with respect to the environment, overlaid onto a rotated onboard camera view. They were successful in improving flying, and even people who were not able to complete the course using the onboard view alone were able to complete it with the chase view.

## 2.2 Teleoperation System Using Past Image Records

Shiroma et al. [14] introduced a Teleoperation System Using Past Image Records (SPIR), a past-image view technique for mobile robots. This technology enabled the operator to ascertain the position and orientation of the robot with respect to its environment from a third-person (bird's-eye view) perspective using only camera and pose information. A system overview is shown in Figure 2.2.

Images taken by a raised camera were saved in a buffer along with their position and the time at which they were taken. No images are stored while the robot is not moving. Three evaluation functions were proposed to select an optimal image to use as the background:

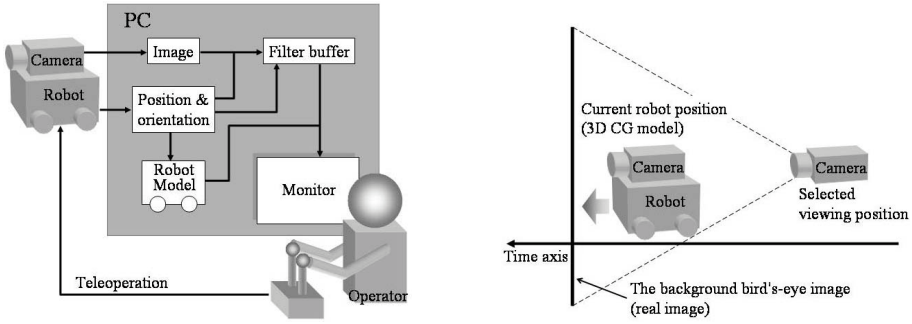


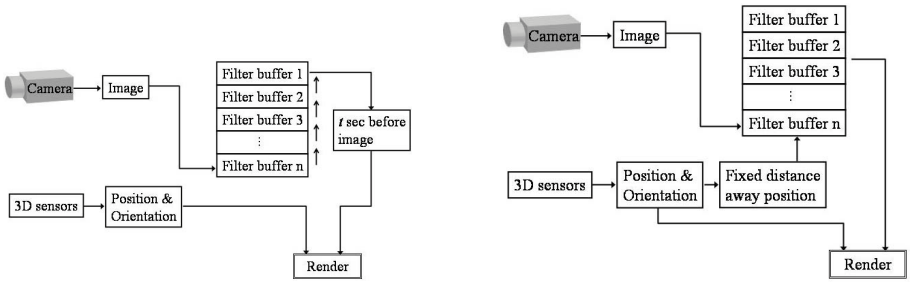

Figure 2.2: System overview of SPIR.[14]

**Fixed time delay**: Use images taken a certain period earlier.

**Fixed distance**: Use images taken at approximately the given distance.

**Field of view (FOV) evaluation**: Select the closest image if the robot is in the field of view.

Figure 2.3 shows the algorithms for fixed time delay and fixed distance.

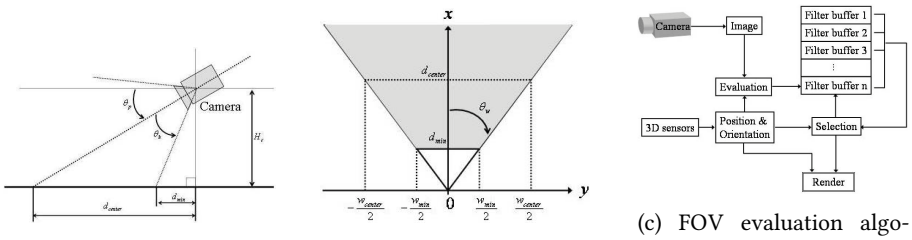



(a) Fixed time delay algorithm.

(b) Fixed distance algorithm.

Figure 2.3: The naïve algorithms.[14]

FOV evaluation uses the fixed-distance algorithm as the base, but also calculates view and centreline distances, as well as the widths at the bottom and centre of the image. The definition of the distances, as well as the algorithm, are shown in Figure 2.4.




(a) Side view.

(b) Top view.

(c) FOV evaluation algorithm.

Figure 2.4: FOV evaluation.[14]

An optimal image was then found using an evaluation function considering several constraints, including FOV and the relative positions of the camera and robot. The latest image is used if no good solution can be found.

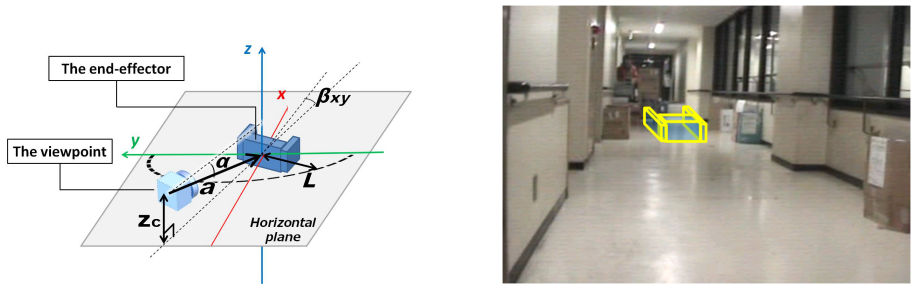
If the bandwidth is low, old images can be reused, updated by an array of position data sent at a higher speed than the video.

### Further development

Sugimoto et al. [15] demonstrated the effectiveness of SPIR with user experiments. Ito et al. [16] decreased choppiness by proposing wider FOVs, and zooming into the background and model as the robot went further into the image plane. They also experimented with adding trajectory forecasting to unmanned ground vehicle interfaces. Screenshots from Ito et al. [16] are shown in Figure 2.5.

Figure 2.5: Zoom with bird's-eye view synthesis.[16]

Murata et al. [17] extended the concept to 3D when they applied it to mobile manipulators. They showed a significant decrease in operators' position error in the vertical direction with a similar error in the longitudinal direction. This system is shown in Figure 2.6.




(a) Geometric relationships.

(b) Basic display concept.

Figure 2.6: SPIR used with a mobile manipulator.[17]

## 2.3 Problems of applying SPIR to quadrotors

When a quadrotor flies forwards, the craft pitches down. For cameras which do not offer gimballing or a wide FOV, though, this means that the horizon could go out of frame. As a result, older video would need to be used, which would likely be from

a longer distance. If the drone turns while going forward, the only usable images might be the live ones.

One potential mitigating factor would be to set a maximum pitch angle, though that would limit the speed of the drone. Even so, during straight and level flight, an operator would only be able to see the drone from behind, unless an elevated camera is attached.

Another possible solution would be to tilt the camera upwards. This would limit the amount of ground it would see during a hover. In addition, it would cause the camera to be pointing at the sky while the drone is outdoors and moving backwards, removing potential reference points.

Finally, localization can be difficult too. Even outdoors, Global Positioning System (GPS) only tends to be accurate to within a few metres, which is much larger than the widths of most drones. While estimates of acceleration can be obtained by modelling the motor outputs and using built-in accelerometers and gyroscopes, there is often drift from wind, as well as significant vibration even during hovering. Integrating acceleration twice to obtain position would cause exponentially increasing drift.

External sources could be used to reduce the error, such as motion capture cameras if indoors, binocular cameras, visual odometry, GPS, or Simultaneous Localization and Mapping (SLAM). That might require the use of more additional tools, such as ultrasonic distance sensors, variometers, or laser range finders. An Extended Kalman Filter (EKF) could then be used to estimate the best location and minimize drift. However, that is beyond the scope of this research.

Some SLAM algorithms, such as Parallel Tracking and Mapping (PTAM), Real Time SLAM (RT-SLAM), or Large-Scale Direct Monocular SLAM (LSD-SLAM), are able to use monocular cameras, but rotation on the spot often causes them to lose keypoints, making them a bad choice for drones.

# Part II

# Implementation

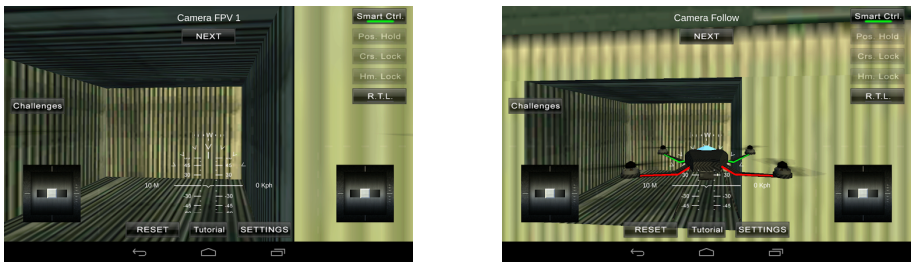
# 3 Proposal

The proposal was to upgrade SPIR to allow it to work better with quadrotors. Operator workload should decrease in positioning and orientation tasks.

A framework should be built which would allow the modular implementation of the core functions of SPIR:

1. Obtain the robot position by the best method for each robot.
2. Save images obtained by the camera, and the pose and time at which they were taken.
3. Select the best frame according to an evaluation function. That robot should ideally be seen from a good position and orientation with respect to the rest of the environment.
4. Generate and display the rendering of the current robot position onto the selected frame.

The system was built in Robot Operating System (ROS) for interoperability and future expansion, as well as its ease of modularization. An example of the difference between first person view and third person view is shown in Figure 3.1.




(a) First-person View.

(b) External view.

Figure 3.1: The difference between FPV and an external view. These screenshots are taken from the QuatcopterFX Simulator app on Android.

The actual implementation is discussed below.

# 4 System overview

The system overview for Subimposed Past Image Records Implemented for Teleoperation (SPIRIT) is shown in Figure 4.1.

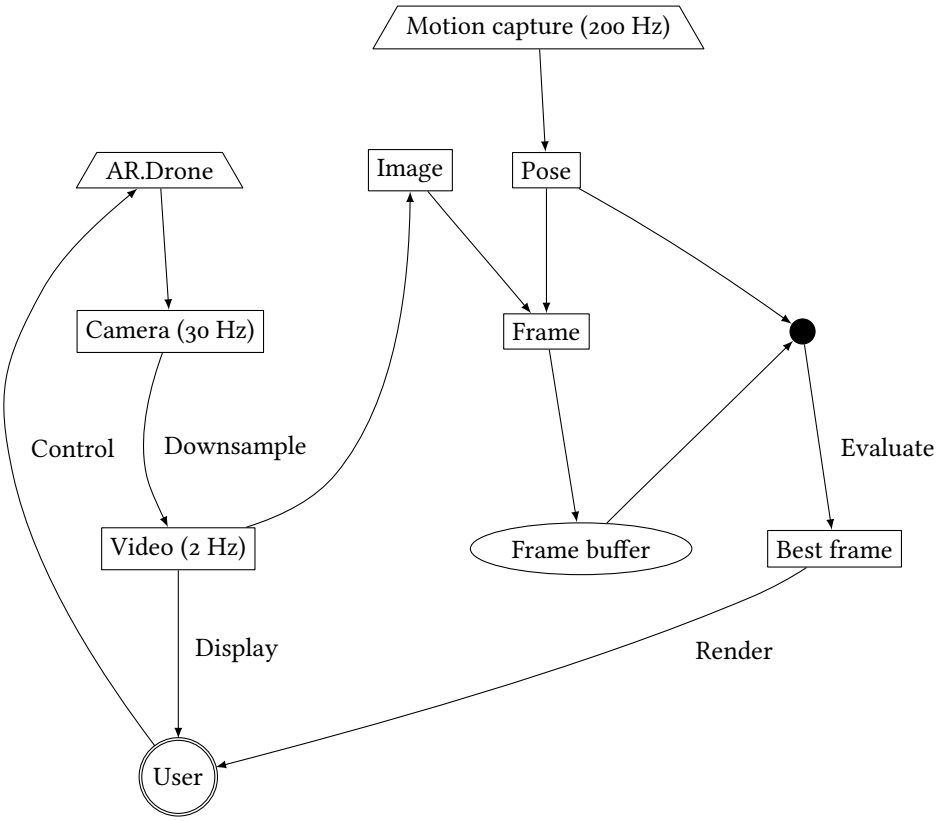



Figure 4.1: SPIRIT system overview.

There are six main parts to the project:

- An operator-controlled gamepad sends commands to the AR.Drone.
- The pose, consisting of the position and orientation of the drone, is calculated by a commercial motion capture system and relayed to the operating station.

- The drone streams video at 30 Hz, which is downsampled to 2 Hz to simulate loss of signal (LOS) conditions.
- The 2 Hz video feed is shown to the operator without modification if the option is selected.
- All received images are also stored in a chronological array. When a new pose is received, it is checked against all frames using an evaluation function.
- The frame with the lowest score is selected, and has the current pose rendered onto it to show to the operator.

# 5 General components

## 5.1 Environment

The operating station runs in a single Docker[1] container on a Linux laptop. The primary components of the Docker image are:

- ROS
- Python 2.7
- Virtualenv
- OpenGL
- Nvidia drivers
- Necessary ROS and Python packages

A separate Windows laptop was connected to the lab's motion capture system and to the operating station via ethernet. The AR.Drone connects to the operating station by acting as a Wifi access point. Communication with the drone is achieved using the ardrone_autonomy package from Simon Fraser University's Autonomy Laboratory.

The research code was written entirely in Python. Taking advantage of ROS's modularity, each function is completely encapsulated in one or more nodes. As a result, implementing a new algorithm or introducing a new method to obtain data simply involves writing new code which publishes to the correct nodes. Any supported programming languages can be used. For instance, the speed of C++ can be leveraged where necessary.

In addition, launch files are generated on the fly from xacro files, with configuration data stored in various Yet Another Markup Language (YAML) files. Selecting and changing coefficients of the various evaluation functions, for example, is in the launch_config.yaml file. The configuration generator inspects the source code to ensure that the evaluation function class exists and that each component is indeed a runnable method. The generator also sets the default values of various parameters, such as the AR.Drone's Internet Protocol (IP) address or the device file for the controller. Identifying the controllers and selecting the correct mapping has also been demonstrated, but is not implemented in the final setup.

The ROS graph is shown in Figure 5.1.

[1] Docker is an open-source software project which automates the deployment of Linux applications

## 5.2 Control

The operating station receives operator input from an off-the-shelf PlayStation 3 (PS3) console gamepad, and forwards it to the drone. After the setup of the button and axis mappings was complete, the controller publishes directly to the relevant `/ardrone` nodes set up by ardrone_autonomy.

There is no automation in flying the AR.Drone apart from the built-in AI. The default control system implemented by Parrot is used.

## 5.3 Video

A live 30 Hz feed is received from the drone. In order to simulate LOS conditions, the feed frequency was dropped to 2 Hz by only selecting every fifteenth frame.

The slower feed is republished for other nodes. Specifically, its frames are the ones stored in the buffer used in selecting past images, and the video was also displayed to the user for increased situational awareness.

## 5.4 Abstractions

ROS's `PoseStamped` messages are wrapped as a `Pose` object, which exposes the position and orientation of the message as numpy arrays. This class also allows the calculation of relative Euler angles and distances with other `Poses`.

New `Images` (from sensor_msgs.msg) and the latest available `Pose` are stored in a `Frame` object. A `Frame` also allows easy calculation the relative position with another `Pose` in the local reference frame. Since the same calculation is run multiple times per frame, the result was memoized, causing a significant speedup.

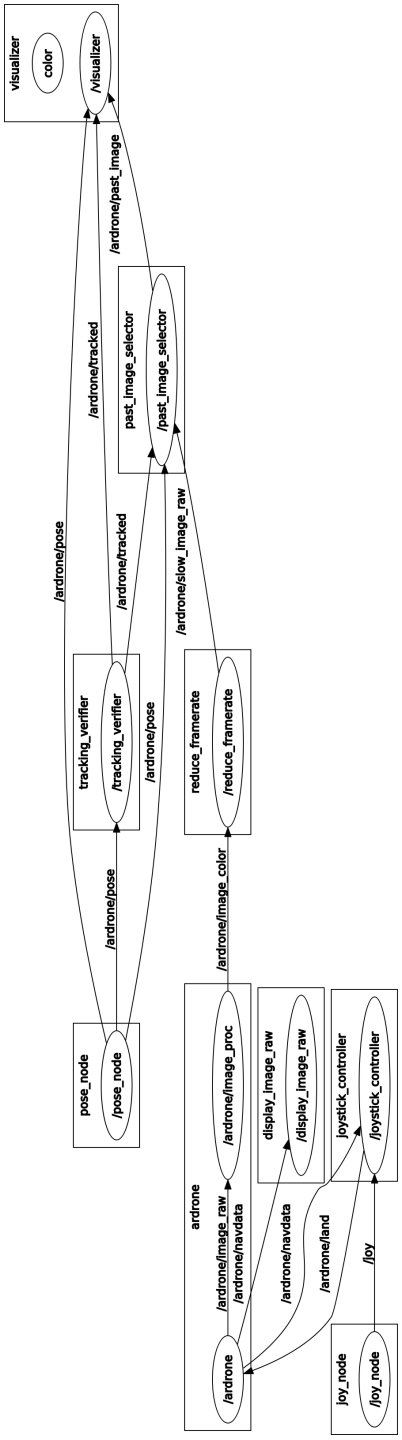
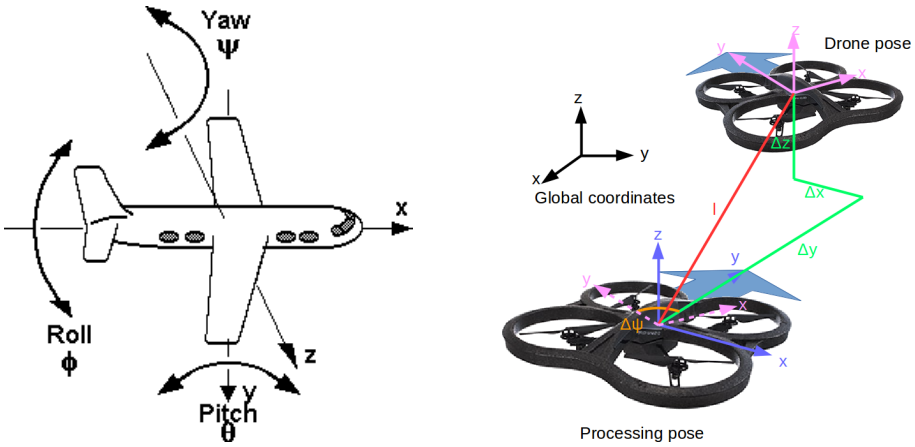



Figure 5.1: The ROS graph for the system.

# 6 Poses

## 6.1 Terminology

Roll ($\phi$), pitch ($\theta$), and yaw ($\psi$) refer to rotations about the respective axes ($x_b$, $z_b$, $y_b$), as shown in Figure 6.1a. The convention used in this report are with respect to the body frame of reference. The positive direction is determined by the right-hand rule.




(a) Roll, pitch, and yaw.

(b) Pose variables.

Figure 6.1: Figure 6.1a shows the definitions of roll, pitch, and yaw with respect to an aircraft in the body frame of reference. Figure 6.1b shows the pose variables, and the relative positions of the current drone position and the frame being processed.

The position of the drone is represented by $\mathbf{x}$, and defined as:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{6.1}$$

where $x$, $y$, and $z$ are the $x$, $y$, and $z$ components.

The orientation of the drone is represented by a quaternion, $\mathbf{q}$, and defined in the Jet Propulsion Laboratory (JPL) convention as:

$$\mathbf{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{6.2}$$

such that the quaternion can also be represented as $q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$. $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are the basis elements of the quaternion, and they are related by:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \tag{6.3}$$

The state vector is $\mathbf{a} = \begin{bmatrix}\mathbf{x} & \mathbf{q}\end{bmatrix}^\top$.
The rotation matrix $\mathbf{R}$, based on quaternions, is defined as: [18]

$$\mathbf{R} = \begin{bmatrix} 1 - 2{q_y}^2 - 2{q_z}^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z + 2q_yq_w \\ 2q_xq_y + 2q_zq_w & 1 - 2{q_x}^2 - 2{q_z}^2 & 2q_yq_z - 2q_xq_w \\ 2q_xq_z - 2q_yq_w & 2q_yq_z + 2q_xq_w & 1 - 2{q_x}^2 - 2{q_y}^2 \end{bmatrix} \tag{6.4}$$

A quaternion can be converted into Euler angles using: [19]

$$\begin{bmatrix}\phi \\ \theta \\ \psi\end{bmatrix} = \begin{bmatrix} \arctan \frac{2(q_wq_x+q_yq_z)}{1-2({q_x}^2+{q_y}^2)} \\ \arcsin\left(2\left(q_wq_y - q_zq_x\right)\right) \\ \arctan \frac{2(q_wq_z+q_xq_y)}{1-2({q_y}^2+{q_z}^2)} \end{bmatrix} \tag{6.5}$$

Finally, $C$, $D$, $F$, and $G$ are the reference frames of the currently displayed frame, the drone, the frame being processed, and the global coordinate system respectively.

## 6.2 Relative poses

### Relative position

In order to find the relative position of the drone with respect to the frame being processed, the orientation of the frame must first be cancelled out by the rotation matrix.

$${}^F_D\Delta\mathbf{x} = {}^G_F\mathbf{R}\left(\mathbf{x}_D - \mathbf{x}_F\right) = \begin{bmatrix}\Delta x \\ \Delta y \\ \Delta z\end{bmatrix}_{FD} \tag{6.6}$$

### Relative orientation

In order to find the relative orientation of the drone with respect to the frame being processed, we need to find the quaternion product.

$${}^F_D\Delta\mathbf{q} = \hat{\mathbf{q}}_F\hat{\mathbf{q}}_D \tag{6.7}$$

First, set the imaginary components of the quaternion to $\mathbf{I}$ such that:

$$\mathbf{I} = \begin{bmatrix}q_x \\ q_y \\ q_z\end{bmatrix} \tag{6.8}$$

and $\mathbf{q} = \begin{bmatrix}q_w & \mathbf{I}\end{bmatrix}^\top$. The solution is:

$${}^F_D\Delta\mathbf{q} = \begin{bmatrix}q_{wF}q_{wD} - \mathbf{I}_F \cdot \mathbf{I}_D q_{wF}\mathbf{I}_D + q_{wD}\mathbf{I}_F + \mathbf{I}_F \times \mathbf{I}_D\end{bmatrix} \tag{6.9}$$

## 6.3 Motion capture

Optitrack's *Motive* software is used to collect motion capture (mocap) data on the position and orientation of the drone. Four infrared markers are set onto the top of the drone, and the Optitrack infrared cameras are connected to the mocap computer. The markers forming the drone are defined as a rigid body, allowing the pose information to be calculated for the entire drone at once. The interface is shown in Figure 6.2

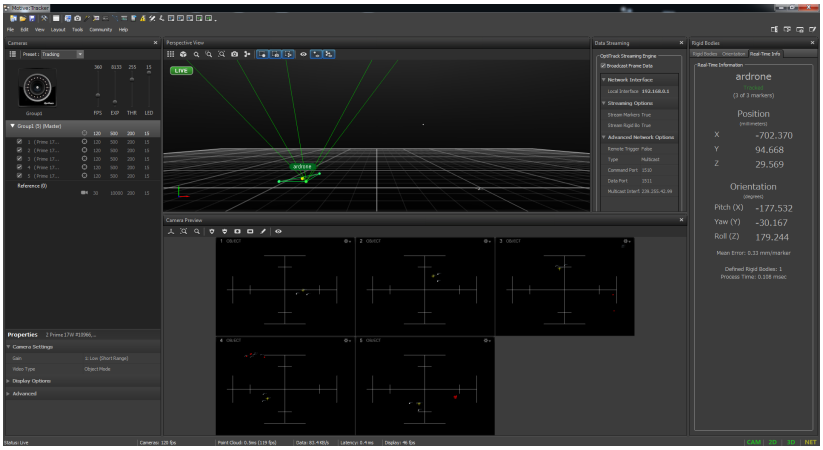



Figure 6.2: The motion capture interface.

The mocap system then streams the data, including the rigid body information, via multicast. The data is received by the operating station, interpreted by ROS's mocap_optitrack package, and republished to `/ardrone`'s `pose` and `ground_pose` nodes. The latter uses quaternions instead of the Eulerian coordinates which *Motive* displays.

The motion capture setup was good for prototyping without needing to turn on the drone, and it had good accuracy compared to, say, GPS. However, depending on it led to an extremely limited usage area, since the drone had to be within range of multiple mocap cameras.

If the mocap system is unable to determine the pose of the drone, whether by marker occlusion or lack of information, *Motive* sends the last known position instead of raising an error. In that case, the visualization system would be using out-of-date pose data, which could cause problems. As a result, the pose is tracked over time, and tracking is considered lost if two poses are identical except for the timestamp. The boolean status is published to `/ardrone/tracked` and can be used by other nodes. For instance, if tracking is lost, an audible horn sounds, accompanied by a visual warning for the operator, as shown in Figure 6.3.

## 6.4 Other methods

One method of obtaining the pose, which is not used in the final setup, is the ability of ardrone_autonomy to obtain drone odometry from the bottom-facing camera, IMU, altimiter, and barometer. The ground velocity is calculated using integration of the acceleration as well as optical flow. While the final result proved to be accurate, the


(a) The onboard interface.

(b) The SPIRIT interface.

Figure 6.3: Screenshot of the onboard and SPIRIT interfaces when tracking has been lost.

calculations required were too slow in a tight loop when running on the operating station. Therefore, it was discarded in favour of the motion capture system.

# 7 Selection of past images

Each new image that arrives while the pose is being tracked is automatically packaged into a `Frame` object containing the time it was taken, as well as the position and orientation of the drone at that time. These `Frames` are then stored in a chronological list, which is implemented as a double ended queue.

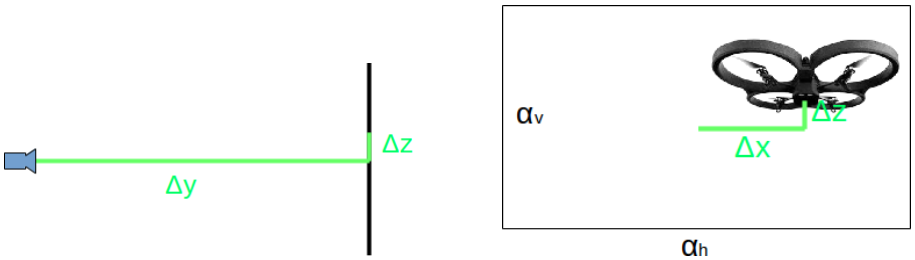
When a new valid pose arrives, it is checked against the frames using an evaluation function. The `Frame` with the lowest score is published for use by the visualization system.

However, while the pose was sampled at 200 Hz (every 5 ms), each iteration of some of the evaluation functions requires up to 0.3 ms to run. As a result, when about twenty frames are collected, a substantial lag in image selection can be detected. To mitigate this, when a new frame is still in the process of being selected, the visualizer displays the new pose on the currently shown frame.

The maximum queue length can be set in the launch parameter configuration file. The final setup used a maximum of 30 frames, or 15 s of video. This allows each loop to run its course before the next pose is received.

## 7.1 Evaluation functions

In this section, all $\Delta$ differences are from the drone with respect to the frame being processed, unless otherwise specified. In addition, $_{ref}$ indicates a reference, or ideal value, which can be set by the operator. So, for instance, $\|\Delta\mathbf{x}_{ref}\|$ is the reference distance at which to keep the drone. Figure 7.1 shows the relative positions. For more information on the notation, please see Section 6.




(a) From the side. (b) From the front.

Figure 7.1: The relative position of the drone with respect to the frame being processed, in terms of camera positioning.

An evaluation function is defined as:

$$E = \sum_{i=1}^{n} k_i f_i \tag{7.1}$$

where $n$ is the number of elements in the evaluation function, $f_i$ is an element, and $k_i$ is its corresponding coefficient. After evaluation, the frame with the lowest score is selected, and its contents are streamed to the `/ardrone` namespace's `past_image` and `past_pose` nodes.

### Evaluation functions in previous works

The simplest evaluation function for past image view is one with constant time delay, as suggested by Shiroma et al.[14] One can quickly find a frame with a specific age by doing a binary search in the chronologically-ordered array. However, while it works well for robots moving at a relatively constant velocity with minimal direction change, it does not track sudden movements, accelerations, or stoppages.

On the other hand, Shiroma et al.'s constant distance evaluation function[14] maintains the distance effectively, and works well with robots which change their orientation gradually. With a drone, though, the operator can move in any direction at any time, including backwards, and the selected frame might not contain the drone's current position at all.

Murata et al.'s evaluation function was designed for 3D environments,[17] and considered the height above the viewpoint as well as the azimuth and tilt angles.

Note that all three evaluation functions have also been implemented, and are accessible by setting the appropriate value in the `launch_config.yaml` file.

### SPIRIT evaluation function

The final version of the SPIRIT evaluation function is:

$$\begin{aligned} E_{\text{SPIRIT}} = {} & k_1 \frac{\sqrt{\Delta x^2 + \Delta z^2}}{\|\Delta \mathbf{x}\|_{ref}} + k_2 \Delta\psi^2 + k_3 \left( \frac{\|\Delta \mathbf{x}\| - \|\Delta \mathbf{x}\|_{ref}}{\|\Delta \mathbf{x}\|_{ref}} \right)^2 \\ & + k_4 {}^{C}_{F}\Delta\psi^2 + k_5 \frac{\left\| {}^{C}_{F}\Delta \mathbf{x} \right\|}{\|\Delta \mathbf{x}\|_{ref}} \end{aligned} \tag{7.2}$$

The five components are as follows:

- $f_1$ represents the centrality, or how close the drone is to the centre of the frame.
- $f_2$ represents the difference in yaw between the drone and the frame. We would like to see the drone directly from behind, and face in the same direction as it.
- $f_3$ represents the distance to the drone. We want to be as close as possible to an ideal distance away, such that the drone is neither too big nor too small.
- $f_4$ represents the difference in yaw between the currently displayed frame and the frame being processed. The smaller it is, the smaller the angular difference is if the frame is selected.

- $f_5$ represents the difference in distance between the currently displayed frame and the frame being processed. The smaller it is, the closer the two frames are and the smoother the transition.

In order to find good coefficients, the video and data of several drone flights were recorded. Then, regions of stability were found by brute forcing simulations with each coefficient ranging from 0 to 10. The results were evaluated based on measures such as frequency of new frame selection. The remaining combinations were checked manually, and potential candidates were flown. The values of the coefficients as used in the experiments are shown in Table 7.1.

Table 7.1: The values of the coefficients used in the user experiments.

| Coefficient | Value |
|---|---|
| $k_1$ | 4 |
| $k_2$ | 2 |
| $k_3$ | 8 |
| $k_4$ | 1 |
| $k_5$ | 2 |

In addition, the reference distance was set to 2.5 m. The thresholds for distance and yaw, below which a new frame is not added to the buffer, were set to 25 cm and 10 degrees respectively.

# 8 Visualization

Visualization in ROS is usually done with rviz. However, due to a lack of good documentation during the early implementation phase, as well as various problems in displaying the background correctly with respect to the drone avatar, it was abandoned for a custom solution. The most advanced state with rviz is shown in Figure 8.1.

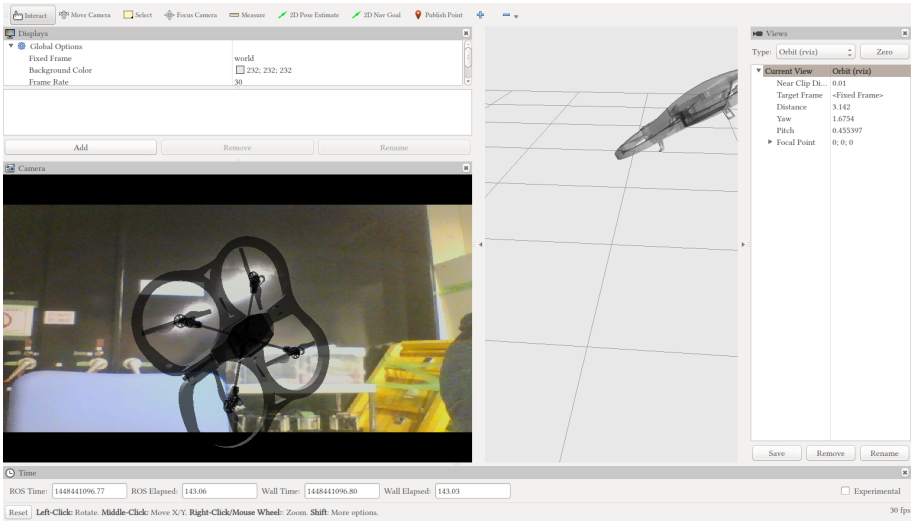



Figure 8.1: Maximum progress with rviz.

## 8.1 Components

Visualization in this project uses three different libraries: PyOpenGL, Pygame, and PyQt5.

### PyOpenGL

PyOpenGL is used to render the avatar of the AR.Drone in the scene. The current drone pose information is in the `/ardrone/pose` node, while the frame information is obtained from the `past_image` and `past_pose` nodes of the same namespace.

Context managers for commonly-used PyOpenGL functionality were written to ease development.

### Pygame

Pygame is used to produce the horn sounds if tracking is lost or reacquired. It is also used to host the PyOpenGL scene which is rendered. However, simple tasks require a roundabout way of doing things, so future iterations may not use Pygame at all.

### PyQt5

PyQt5 is used to create a window for streaming video feeds. When the AR.Drone is connected, its remaining battery percentage and flight status are displayed in the status bar at the bottom of the window.

PyQt5 could also be used to replace Pygame for producing the horn sound when tracking is lost or reconnected, as well as hosting the PyOpenGL scene. At the time of writing, this is yet to be done.

## 8.2 Rendering

The visualizer is run as a multithreaded application in order to maximize responsiveness. After obtaining the relevant data, the relative positions and orientations of the drone and the frame are calculated. A new texture is automatically loaded whenever a new `past_image` is published, and is used as the frame when the scene is rendered.

OpenGL uses an axis-angle representation, so the relative orientation is converted from quaternion to axis angle using the following formula:

$$\theta = 2 \arccos q_w \tag{8.1}$$

$$\mathbf{e} = \begin{cases} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, & \text{if } \theta = 0 \\ \begin{bmatrix} q_x & q_y & q_z \end{bmatrix}, & \text{if } \theta \bmod 180^\circ = 0 \\ \dfrac{\begin{bmatrix} q_x & q_y & q_z \end{bmatrix}}{\sqrt{1 - {q_w}^2}} & \text{otherwise} \end{cases} \tag{8.2}$$

where $\theta$ is the rotation angle.

The scene is re-rendered whenever a new `pose` is published. First, the relative position is calculated as ${}^{D}_{F}\Delta\mathbf{x} = \mathbf{x}_F - \mathbf{x}_D$, and the orientations of the frame and the drone are converted into an OpenGL quaternion using the following mapping:

$$\begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix} \mapsto \begin{bmatrix} q_w & -q_x & q_z & q_y \end{bmatrix} \tag{8.3}$$

In a new matrix, the perspective is rotated by the same amount the frame is. Since the $q_z$ axis is with respect to the origin, not the camera, that component is first multiplied by $-1$. Next, the relative positions are converted to the OpenGL coordinate frame using the following mapping:

$$\begin{bmatrix} \Delta x & \Delta y & \Delta z \end{bmatrix} \mapsto \begin{bmatrix} \Delta x & -\Delta z & -\Delta y \end{bmatrix} \tag{8.4}$$

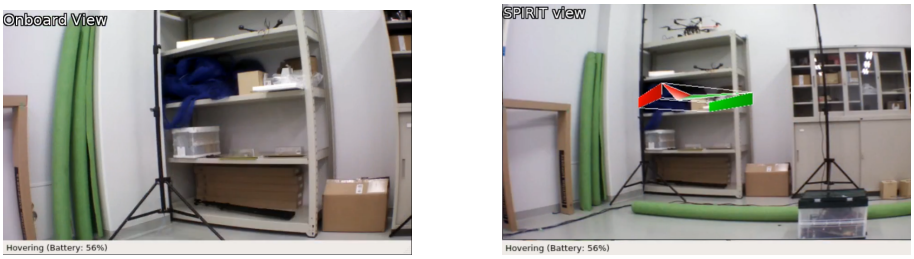
The scene translates by that amount. The current texture is then drawn as a 2D orthographic projection covering the entire scene.

After that, the drone is drawn at the origin. The model of the drone is shown as a blank parallelipiped using OpenGL primitives, with the same dimensions as the edges of the actual drone. A surface in the shape of an arrow is added to the top side,

and coloured using the standard navigation light colouration. Specifically, the left side of the drone is shown in red, while its right side is in green. To draw it, a new matrix is pushed, and the scene is rotated by the relative orientation, after which the individual components are added.

Finally, text can be superimposed using orthographic projection, if it is necessary. This is used, for instance, when warning the operator that there is no data yet, or when tracking is lost.

The final interface is shown in Figure 8.2.




(a) The onboard interface.

(b) The SPIRIT interface.

Figure 8.2: Final interfaces for the onboard and SPIRIT interfaces, with the AR.Drone tracked. Note that the target appears only in the SPIRIT view.

An offline testing mode, with an optional web camera feed, was added to the past-image visualizer for debugging purposes.

# Part III

# Experiments

# 9 Experiment Design

## 9.1 Objectives

The experiments were conducted in order to evaluate the efficiency and efficacy of SPIRIT with respect to a conventional onboard camera in an environment with low communication bandwidth. These were evaluated using both objective and subjective measures, as described in Section 9.4. Particularly of interest were the spacial accuracy with respect to a target, and the time required to complete the task.

## 9.2 Participants

A total of nine people participated in the study. All were volunteers, male, and students of Kyoto University. The youngest was 22 years old, and the oldest was 29. The mean age was 24.2 years old, and the standard deviation was 2.1 years.

Information was collected about their experiences. Three had some experience flying teleoperated robots, of which one had some experience with quadcopters. No participants had any other flying experience, including flight simulators.

An information pamphlet was prepared, and written consent was obtained from each in order to comply with ethical obligations.

## 9.3 Experiment setup

The participants were asked to fly a drone from a starting point to a target 6 m away. The target itself (Figure 9.1) was a raised box 52.5 cm wide, 37 cm deep, and 34 cm high. When they believe that they have arrived, they would indicate as much by pressing a button on the controller, before landing safely. The setup of the flying area can be seen in Figure 9.2.

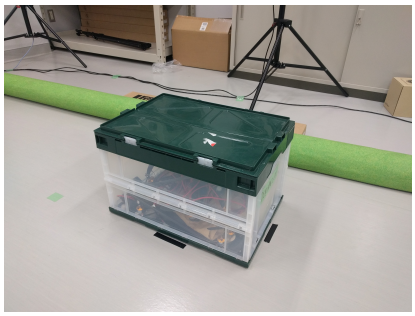


Figure 9.1: The target which users aim towards.

The drone itself was not always stable, and sometimes changed heights or rotated significantly upon takeoff. As a result, the takeoff pad was visible to the operators to minimize the potential for damage to equipment or injury to others. The drone would no longer be visible after moving a short distance towards the target. A closeup of the drone on the launchpad is shown in Figure 9.3.

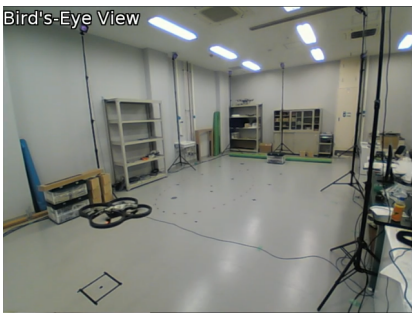



Figure 9.2: The flying area which the experiment was set in.

Figure 9.3: The drone on the launchpad, looking towards the target.

### Timeline

In order to reduce the effect of learning on performance, the order of the trials was changed based on the participant number. Starting from zero, all five even-numbered participants flew with the onboard view twice before subsequently flying twice with SPIRIT. The four odd-numbered participants started with SPIRIT, and then completed the onboard task.

At the beginning of each set of experiments, a verbal explanation was provided to the participant by the experimenter, and consent was obtained.

After that, each user had the opportunity to practice flying the drone for a maximum of five minutes. This practice could be ended prematurely at the request of the operator. During practice, both display interfaces were shown at the same time. There were also no obstructions, so that the user could see the drone all the way to the target, and get a sense of how the correct position and orientation would look like with each system.

After a brief practice session, each person performed one task until the arrival button presses have been received in two runs. They would then perform the second experiment. A workload assessments and a survey were completed after each pair. Therefore, each subject had four successful runs, and performed two workload assessments and two surveys. Some performed a larger total number of experiments, but experienced failures usually due to the drone moving up or stopping without user input. These failures were not considered.

### Trials

At the start of each trial, a bird's-eye view and both interface outputs were recorded, and a custom program recorded relevant data to ROS Bag files. During any given experiment, the participant was only able to see the relevant interface, while the other was hidden. A still from one of the recorded videos is shown in 9.4.

Some changes were made to the onboard view to reduce external influences on the results. First, while the drone provides video at 30 Hz, the experiment simulates an area with a bad connection. The transfer rate was artificially reduced to 2 Hz, which is the same frequency used by SPIRIT to capture frames. In addition, a loud warning sound used to notify the operator of loss of position tracking was supressed during trials using the onboard view in order to reduce distractions to the operator.


Figure 9.4: A still from a recorded video.

On the other hand for SPIRIT trials, no computer graphics (CG) model was displayed at takeoff because no images had yet been captured of the position. Upon launch, then, the user would instead move the drone about one body length forward in order to cause the CG model to show. After that, operation would proceed normally until landing.

## 9.4 Evaluation method

To analyze the efficiency of the proposed interface, all ROS topics were recorded and studied. The evaluation can be broken down into five groups, as shown below.

**Accuracy** Accuracy was calculated by using the position of the drone when the arrival button was pressed. The data was analyzed by the differences in $x$ and $y$, as well as the absolute distance from the centre of the target.

**Path length**: The path length was calculated by adding the distances between points. Also calculated where the total amount of motion in one axis, and in one direction.

**Duration**: Since the amount of time taken from initalization until takeoff varied, the duration of the task was calculated from takeoff to the time of the first arrival button.

**Workload**: The self-reported cognitive workload of the operator is measured by using the NASA Task Load Index (NASA-TLX), developed by National Aeronautics and Space Agency (NASA). Six components are each rated from 0 to 20, inclusive, and then weighted according to their contributions, with a maximum possible score of 100. The higher the score, the higher the perceived workload.

**Survey**: The participants were asked to complete a survey at the end of each set of tasks, rating their perception on multiple criteria. The survey checked control and understanding of position, orientation, and relative position with the target. The scale of each question was from 1 to 7, with 7 being the best.

Free answers were also collected from willing subjects.

It was hypothesized that the proposed interface would lead to:

- increased accuracy
- shorter path length
- shorter duration
- lower workload
- better user satisfaction

## 9.5 Data analysis

Since the same people were performing a similar task multiple times, and two repetitions were used for each interface, a repeated measurement design was used.

John K. Kruschke's Bayesian Estimation Supersedes the $t$ Test (BEST)[20] was used in order to evaluate the differences in the physical results obtained from each interface. From the paper's abstract:

> Bayesian estimation for two groups provides complete distributions of credible values for the effect size, group means and their difference, standard deviations and their difference, and the normality of the data. The method handles outliers. The decision rule can accept the null value (unlike traditional $t$ tests) when certainty in the estimate is high (unlike Bayesian model comparison using Bayes factors). The method also yields precise estimates of statistical power for various research goals.

BEST was implemented in Python using PyMC3. The sampling algorithm was a Markov Chain Monte Carlo with Metropolis-Hastings. 2000 steps were calculated.

The results of NASA-TLX and the survey were analyzed using a paired $t$ test. Because the population size was small, the effect size was calculated using Hedges's $g$, such that:

$$g = \frac{\mu_1 - \mu_2}{\sigma^\star_{\text{pooled}}} \times \left( \frac{N-3}{N-2.25} \right) \times \sqrt{\frac{N-2}{N}} \tag{9.1}$$

and $\sigma^\star_{\text{pooled}}$, the weighted pooled standard deviation, is:

$$\sigma^\star_{\text{pooled}} = \sqrt{\frac{(n_1 - 1)\,\sigma_1^2 + (n_2 - 1)\,\sigma_2^2}{N-2}} \tag{9.2}$$

# 10 Results

The numerical results of the experiments and the analyses are shown for each experiment type. Their Highest Posterior Density intervals (HPDs) are not shown in the table, but they are mentioned when needed. The difference of means, the probability mass (PM) on one side of zero, and the effect sizes (Hedges's $g$) are shown. As a general rule of thumb, an effect can be considered statistically significant at a desired level if the Credible Interval (CI) does not contain zero, or, equivalently, if the reported PM is greater than 95%. Values of 0.2, 0.5, and 0.8 can be considered small, medium, and large effects respectively.[21]

## 10.1 Path length

Figure 10.1 shows the paths flown by the operators, as well as the locations at which they arrived. The target is the coloured rectangle, while the dashed line represents the area within which one part of the drone would be over the target.

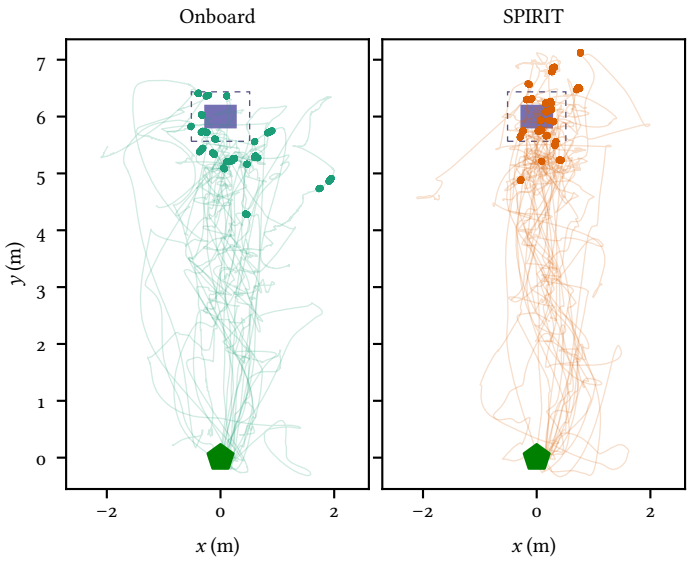



Figure 10.1: Overview of the paths flown by the operators and their arrival locations.

All SPIRIT subjects confirmed that they were able to see the target, as well as their relative position in $x$, but many had difficulty estimating their position in $y$. This can be seen by the amount of paths which overfly the target area completely compared to the onboard view. It seems notable that onboard pilots erred on the side of caution

and undershot their approach. This may be due to the fact that the target disappears earlier when using the onboard view, and, with no depth perception, the user cannot rely on environmental cues.

The tendency of the drone to yaw to the right can clearly be seen, where users initially start out by moving to the right before correcting their path. In addition, SPIRIT users tend to take a more consistent route to their destination, and stay in a more narrow zone. This might indicate that they are more comfortable with the interface.

The path length is shown in Figure 10.2, and the summary is shown in Table 10.1.

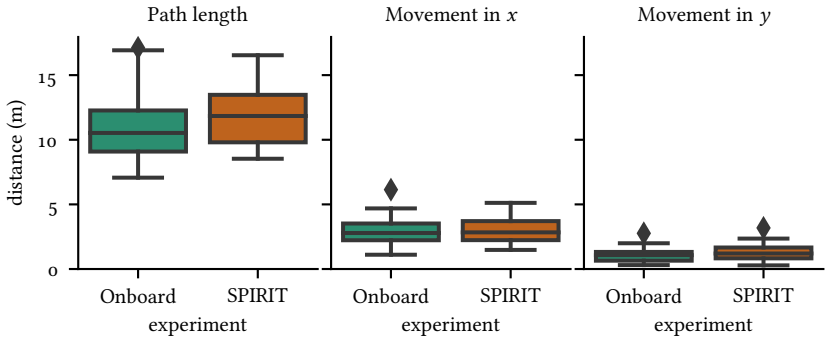



Figure 10.2: Path length, including total movement in $x$ and $y$.

Table 10.1: Summary of the path length.

| | Onboard | | SPIRIT | | | | |
|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\Delta\mu$ | PM | $g$ |
| Path length (m) | 10.849 | 2.727 | 11.914 | 2.706 | −1.049 | 87.0% | −0.390 |
| Movement in $x$ (m) | 2.862 | 1.055 | 2.944 | 1.014 | −0.078 | 58.3% | −0.078 |
| Movement in $y$ (m) | 1.043 | 0.573 | 1.268 | 0.687 | −0.220 | 82.7% | −0.360 |

An ideal run with zero wasted motion would have a total path length of 6.0 m. However, the mean path length for runs using the onboard view was $10.849 \pm 2.727$ m, while that for SPIRIT runs was $11.914 \pm 2.706$ m. That is, SPIRIT users flew 1.05 m longer than their onboard counterparts. It appears that there is a small effect size ($g$=−0.390), but the PM is only 87.0%.

There was almost no difference (−0.078 m) in total motion in the $x$ direction, but there was a small, nonsignificant correlation in $y$ ($\Delta\mu$=−0.220 m, $g$=−0.360, PM=82.7%).

From Figure 10.3, it appears that the first and third SPIRIT flights produced longer path lengths. In both these cases, the operator was using the system for the first time. Towards the end of that first flight, and continuing into their next attempt, the path lengths are comparable to those flown with the onboard view. This may be a statistical aberration, or it could be the effect of familiarity with the system.

## 10.2 Accuracy

Figure 10.4 shows the location of all the arrival points in each of the groups with respect to the target.

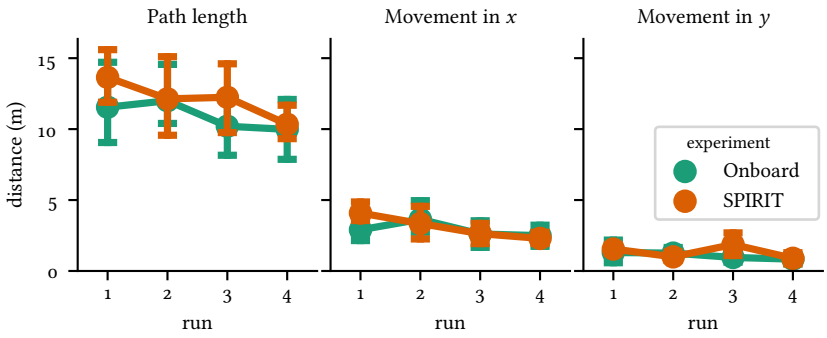



Figure 10.3: The change in path length across runs. The movement in $x$ and $y$ was larger with SPIRIT on the first and third runs, but is similar in subsequent runs.

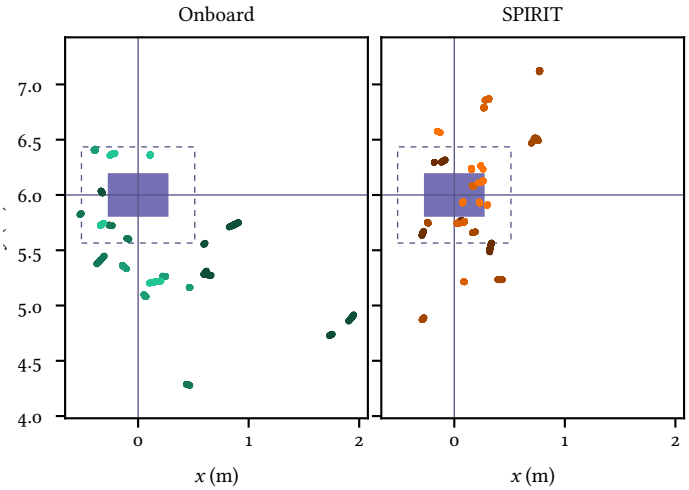



Figure 10.4: Detail of the arrival points with respect to the target. Lighter points represent later runs.

Out of the eighteen arrivals, none of the onboard ones are directly above the target, and only seven had a portion of the drone above the target. The distribution is very wide and not precise, and was slightly semicircular. Some students have commented that they were using a motion capture camera pole as a marker, and orienting themselves around that.

By contrast, SPIRIT operators obtained a much more consistent result, with both higher accuracy and precision. Four out of eighteen were directly above the target, and a further eight had a portion of the drone above the target. This distribution is much more concentrated.

Figure 10.5 shows the distance from the target at the time of arrival. The data is summarized in Table 10.2. As with the paths, people using the onboard view had a strong backward bias, with a mean of −0.471 m. The 95% HPD was between −0.737 and −0.227 m, which is completely outside the target's $y$-region. They also had a large standard deviation in $x$ despite having a mean close to the centre ($\mu$=0.041 m, $\sigma$=0.428 m).

Meanwhile, the SPIRIT view had a strong rightward bias, with a mean of 0.138 m.

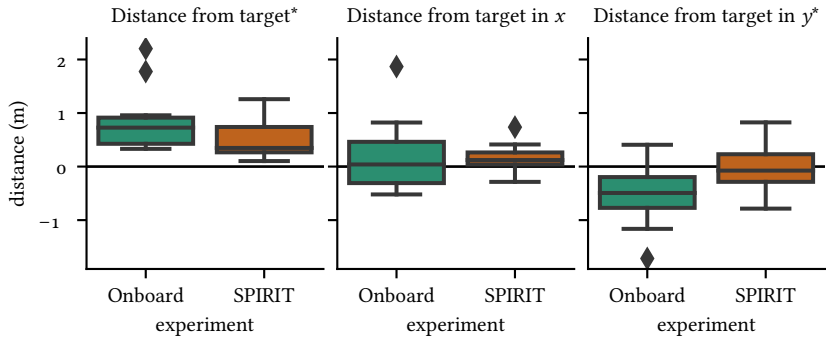



Figure 10.5: Distance from target, including in $x$ and $y$, at the time of arrival.

Table 10.2: Summary of the arrival distance in $x$ and $y$.

| | Onboard | | SPIRIT | | | | |
|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\Delta\mu$ | PM | $g$ |
| $x$-position (m) | 0.041 | 0.428 | 0.138 | 0.170 | −0.100 | 76.2% | −0.345 |
| $y$-position (m)* | −0.471 | 0.445 | −0.071 | 0.364 | −0.396 | 98.3% | −1.000 |

Nevertheless, the 95% HPD of 0.038 to 0.237 m is completely above the target. On the other hand, while the mean of the $y$ error (−0.071 m) is above the target, its large standard deviation of 0.364 m means that the users can be off-target by up to about one target length (95% HPD: −0.293 to 0.149 m).

Root Mean Square Errors (RMSEs) for $x$ and $y$ are shown in Figure 10.6, and summarized in Table 10.3. The total RMSE was almost identical to the total distance shown in Figure 10.5, and are thus treated as one in the analysis below.

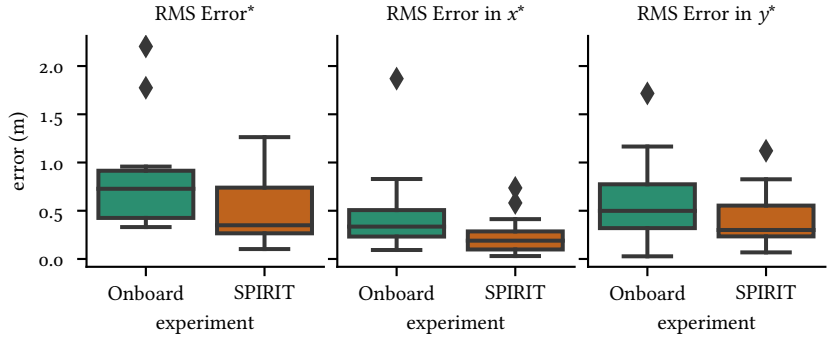



Figure 10.6: RMSE in distance from target, including in $x$ and $y$ at the time of arrival.

While the onboard view had an error of 0.667 ± 0.287 m, SPIRIT had an error of only 0.403 ± 0.237 m. Broken down, the $x$- and $y$-RMSEs for the onboard view were 0.337±0.190 m and 0.521±0.293 m respectively. This compares to just 0.191±0.121 m and 0.339 ± 0.222 m for SPIRIT, respectively.

The difference is significant and the effect is large for the total distance ($\Delta\mu$=0.265 m, $g$=1.059, PM=98.5%) and $\text{RMSE}_x$ ($\Delta\mu$=0.147 m, $g$=0.958, PM=98.0%). They are signifanct and medium, respectively, for $\text{RMSE}_y$ ($\Delta\mu$=0.184 m, $g$=0.737, PM=95.2%).

Table 10.3: Summary of the arrival RMSE.

| | Onboard | | SPIRIT | | | | |
|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\Delta\mu$ | PM | $g$ |
| RMSE (m)* | 0.667 | 0.287 | 0.403 | 0.237 | 0.265 | 98.5% | 1.059 |
| $\text{RMSE}_x$ (m)* | 0.337 | 0.190 | 0.191 | 0.121 | 0.147 | 98.0% | 0.958 |
| $\text{RMSE}_y$ (m)* | 0.521 | 0.293 | 0.339 | 0.222 | 0.184 | 95.2% | 0.737 |

Figure 10.7 shows the change in the RMSEs across runs. It is consistently lower with SPIRIT, apart for the second run, which is slightly higher than the onboard view. This may be an aberration, given the low amount of participants.

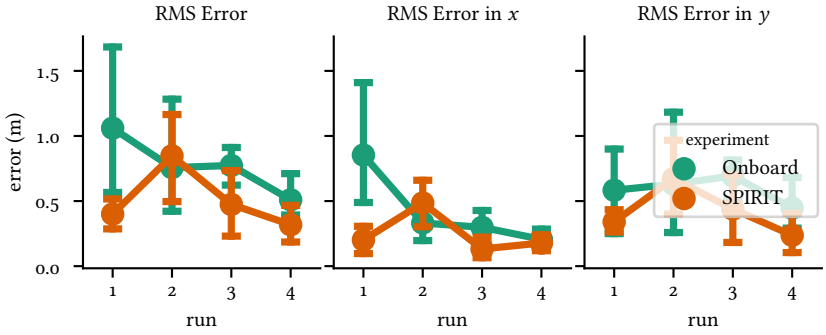



Figure 10.7: Arrival RMSE change across runs.

## 10.3 Duration

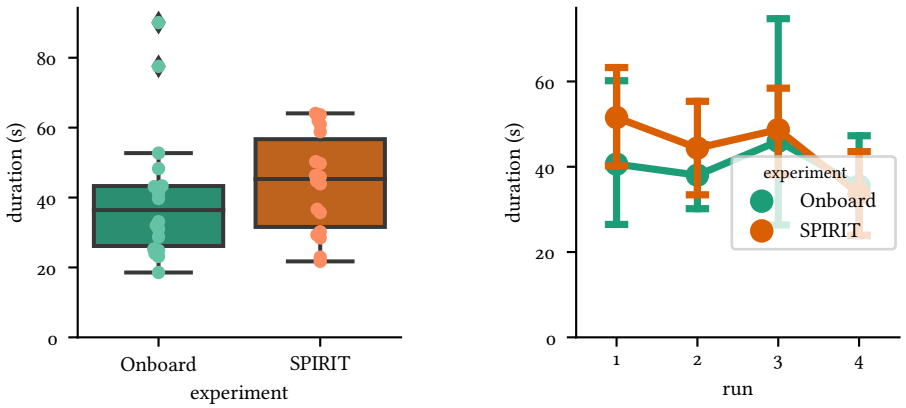
Figure 10.8a shows the duration of each type of experiment, and the data is summarized in Table 10.4. SPIRIT seemed to have a longer duration, but with less spread. In fact, the onboard view has a mean duration of 39.265 s, and a standard deviation of 18.984 s, while SPIRIT has $44.181 \pm 15.140$ s.

This difference, though, is not signficant, and the effect size is small. For duration, $\Delta\mu$=−4.841 s, $g$=−0.294, PM=78.1%.

Table 10.4: Summary of the duration.

| | Onboard | | SPIRIT | | | | |
|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\Delta\mu$ | PM | $g$ |
| Duration (s) | 39.265 | 18.984 | 44.181 | 15.140 | −4.841 | 78.1% | −0.294 |

Looking at Figure 10.8b, the difference between the durations decreased in each run, and SPIRIT was faster by the fourth run. This could be indicative of ease of learning, since the improvements were being made faster than with the onboard version. Further experimentation is needed to verify this hypothesis. For example, if the users are given a longer training session on a different course, they would have already gotten used to the system.




(a) Duration of each type of experiment.

(b) Change in duration across runs.

Figure 10.8: The duration of the flight, from takeoff until the arrival button was pressed.

## 10.4 Workload

Figure 10.9 shows the result of the NASA-TLX survey, both in aggregate and by component. Because the Task Load Index (TLX) responses are subjective, and the scale itself is an ordinal rather than interval scale, no actionable information can be gleaned from this small a sample size.[22] Instead, general trends may be observed.

The six components in Figure 10.9b are, in order:

- **MD**: Mental Demand
- **PD**: Physical Demand
- **TD**: Temporal Demand
- **P**: Performance
- **E**: Effort
- **F**: Frustration

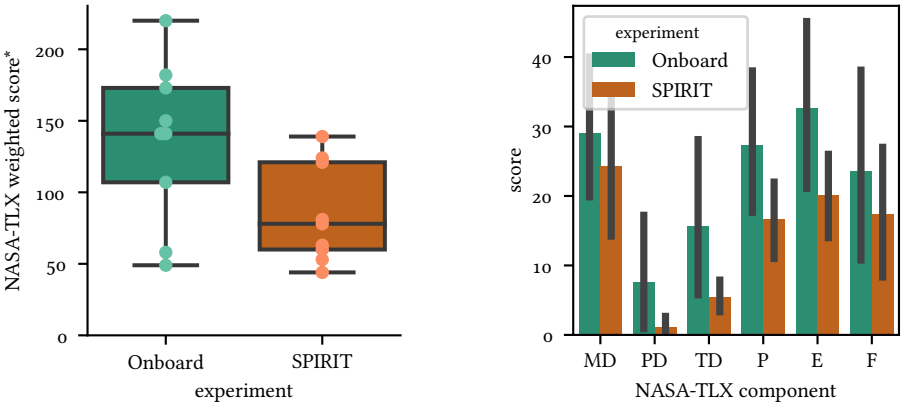
and the weighted score is WS.

Most onboard pilots who had a high score for physical demand mentioned that it was due to the fact that they needed to move in short burts to keep from hitting their surroundings. Since the frame rate was so low, the drone would move a significant distance by the time a frame updated. This raised stress and caused some frustration.

Inherent issues in the system, such as with the drone's tendency to drift right, or the lack of depth perception, also contributed to frustration and mental demand, but it had a similar effect when using either system.

The data is summarized in Table 10.5.

A large, significant reduction of 35.74% in the weighted TLX score was seen. Analysis of the components show that the scores decreased across the board. There was a medium to high effect for physical ($g$=−0.626), temporal ($g$=−0.666), performance ($g$=−0.699), effort ($g$=−0.718), and overall score ($g$=−0.978).




(a) NASA-TLX aggregate results.

(b) NASA-TLX component analysis.

Figure 10.9: Results for the NASA-TLX survey.

Table 10.5: The summary of the NASA-TLX data.

| | Onboard | | SPIRIT | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\Delta\mu$ | $t$ | $p$ | $g$ |
| MD | 29.000 | 15.922 | 24.222 | 17.908 | −4.778 | −1.29269 | 0.23220 | −0.253 |
| PD | 7.556 | 12.885 | 1.111 | 2.261 | −6.444 | −1.81051 | 0.10781 | −0.626 |
| TD | 15.667 | 19.339 | 5.333 | 3.808 | −10.333 | −1.73228 | 0.12146 | −0.666 |
| P | 27.222 | 16.998 | 16.667 | 8.902 | −10.556 | −1.64399 | 0.13880 | −0.699 |
| E | 32.667 | 19.755 | 20.111 | 10.167 | −12.556 | −2.19108 | 0.05982 | −0.718 |
| F | 23.556 | 22.328 | 17.333 | 15.149 | −6.222 | −1.28600 | 0.23441 | −0.293 |
| WS* | 135.667 | 56.214 | 84.778 | 34.662 | −50.889 | −2.77594 | 0.02408 | −0.978 |

## 10.5 Survey

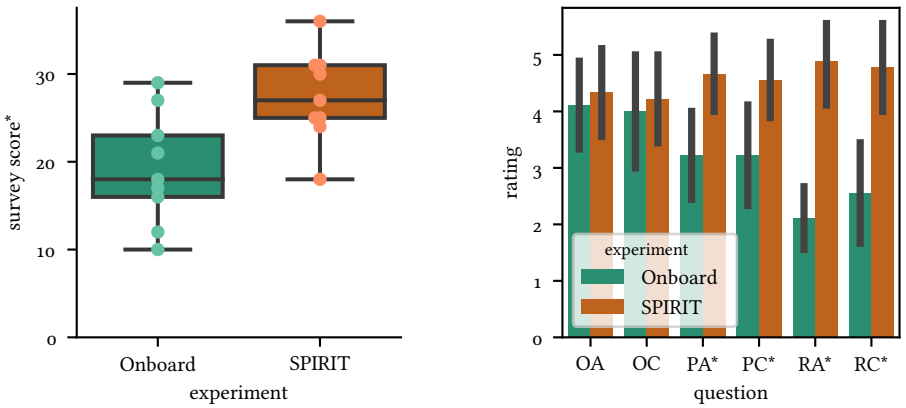
The six components in Figure 10.10b are, in order:

- **OA**: Orientation awareness
- **OC**: Orientation control
- **PA**: Relative positional awareness
- **PC**: Absolute positional control
- **RA**: Relative positional awareness
- **RC**: Relative positional control

and the survey score is SS.

Figure 10.10 shows the result of the survey. Again, scores increased for each category.

The data is summarized in Table 10.6.




(a) Survey aggregate results.

(b) Survey component analysis.

Figure 10.10: Results for the survey.

Table 10.6: The summary of the survey data.

| | Onboard | | SPIRIT | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\Delta\mu$ | $t$ | $p$ | $g$ |
| OA | 4.111 | 1.269 | 4.333 | 1.323 | 0.222 | 0.32552 | 0.75314 | 0.154 |
| OC | 4.000 | 1.581 | 4.222 | 1.302 | 0.222 | 0.29251 | 0.77734 | 0.138 |
| PA* | 3.222 | 1.202 | 4.667 | 1.000 | 1.444 | 2.87122 | 0.02079 | 1.173 |
| PC* | 3.222 | 1.481 | 4.556 | 1.130 | 1.333 | 2.41209 | 0.04237 | 0.909 |
| RA* | 2.111 | 0.928 | 4.889 | 1.054 | 2.778 | 4.85643 | 0.00126 | 2.512 |
| RC* | 2.556 | 1.509 | 4.778 | 1.302 | 2.222 | 3.25515 | 0.01161 | 1.416 |
| SS* | 19.222 | 6.399 | 27.444 | 5.223 | 8.222 | 2.93892 | 0.01874 | 1.264 |

SPIRIT did not affect orientation awareness or control. However, it significantly increased awareness and control for both absolute and relative positioning. The aggregate score also significantly increased.

The largest effect was on the awareness of the position of the drone with respect to the target ($g$=2.512, $p$=0.00126). One strategy that SPIRIT users utilized was slewing to the side in order to get a perspective view of the location of the drone with respect to the target. By extrapolating the vertical edges of the target, they were able to increase their understanding of the situation.

# Part IV

# Conclusions and Recommendations

# 11 Conclusions

SPIRIT is a novel robotics teleoperation system which overlays the current position and orientation of a vehicle onto previously acquired images. This research focuses on developing a SPIRIT-based user interface for aerial robots. The proposed method combines FOV information with state estimation, and selects a suitable image to use as a background. It works even in a low-throughput environment where video data might suffer, and only requires one camera.

Experiments were conducted to study the efficacy at increasing accuracy while decreasing time and wasted movements. Participants were tasked with flying a drone to a location above a known target while using the proposed system, and comparing the results with those obtained by flying with a slowed-down onboard camera.

SPIRIT significantly improved accuracy from 0.666 m to 0.401 m, an improvement of almost 40% ($\Delta\mu$=0.266 m, $g$=1.053, CI=98.1%), and reduced the overall workload by 37.5% ($p$=0.02408, $g$=−0.978), especially in the physical, temporal, and performance metrics. It also significantly improved the users' control and awareness of absolute and relative positioning. Their self-assessed ability to stay above the target increased by 86.9% from an average rating of 2.556/7 to 4.778/7 ($p$=0.00126, $g$=2.511).

There were non-significant increases in time and path lengths, but improvements over subsequent runs indicate that these are simply an issue of familiarity with the system. Further testing is needed to verify the effect of SPIRIT on completion time and path length, and its efficacy in various environments.

Overall, the tests were successful, and all participants gave praise for the system, as well as valuable feedback.

# 12 Future work

Several suggestions for improving the functionality of SPIRIT are presented here. They have been drawn from personal experience as well as user suggestions.

The user experience may be improved if zoom functionality is added. That way, the size of the drone model would remain relatively constant, while the background changes, even with the same image as its base. This was present in the version of SPIR shown in [16].

Similarly, it might be useful to always keep the horizon horizontal and tilt the image by the amount the drone was tilted when the frame was captured, as was implemented by Hing et al.[11] Extending it to all three Eulerian axes would allow, for instance, the pitch to be used more effectively. This would be useful with modern drones, which often use gimballed cameras.

It was discovered that depth perception was difficult when using SPIRIT, and is problematic when moving around obstacles. It might be possible to improve this by showing a shadow where the ground should be. If distance estimation is used, it could be used in a wide variety of environments. Another potential solution is to use binocular cameras and have the user wear a head-mounted display.

Many users mentioned that the motion capture pole near the target was used as a marker to help orient the drone when flying using the onboard view. They felt that it provided an unfair advantage, and that other methods of placing the motion capture camera might make the task much more difficult. Removing the pole might give a fairer comparison with SPIRIT. Alternatively, use other methods of localization to allow the drone to be used in various locations.

Performance can be improved by replacing Pygame with PyQt5. This might also solve the problem with the sound system not being able to initialize, which had previously necessitated restarts.

Finally, using an $n$-tree (an octree, or, its extension, a hextree), to prune the search space might enable a bigger buffer. A hextree would be implemented using $x$, $y$, and $z$ positions, as well as yaw. Higher dimensions for gimballed platforms would also include pitch and roll. Instead of evaluating the entire buffer, only the frames in which the drone would have been visible in the first place can be checked, thereby significantly reducing processing time.

# Part V

# Appendices

# A System Details

## A.1 AR.Drone

The drone used in this research is the Parrot AR.Drone 2.0 Elite Edition. It was released by Parrot SA in 2012. An image is shown in Figure A.1.

Figure A.1: AR.Drone 2.0 Elite Edition

Technical specifications are shown in Table A.1.

Table A.1: The specifications of the AR.Drone.

| Item | Value |
|---|---|
| Resolution | 720 p |
| Framerate | 30 fps |
| Field of view | 92° |
| Connection | Wi-Fi |
| Gyroscope | 3 axles, accuracy of 2,000°/second |
| Accelerometer | 3 axles, accuracy of +/- 50 mg |
| Magnetometer | 3 axles, accuracy of 6° |
| Pressure sensor | Accuracy of +/- 10 Pa |
| Altitude ultrasound sensor | Measures altitude |
| Brushless motor power | 14.5 W |
| Brushless motor speed | 28,500 rpm |
| Weight with indoor frame | 420 g |

## A.2 Motion capture system

The motion capture system is a set of ten Prime 17W cameras from Optitrack. One is shown in Figure A.2, and technical specifications are in Table A.2.


Figure A.2: The Prime 17W motion capture camera.

Table A.2: The specifications of the iBuffalo BSW20KM11 webcam.

| Item | Value |
|---|---|
| Resolution | 1644 × 1088 pixels |
| Framerate | 30–360 fps |
| Latency | 2.8 ms |
| Horizontal field of view | 70° |
| Vertical field of view | 49° |
| Filter | 850 nm band-pass |

## A.3 Motion capture computer

The motion capture computer is a Clevo W255HS, running Windows 7 Enterprise. Its specifications are in Table A.3.

Table A.3: The specifications of the Clevo W255HS computer used to interface with the motion capture system.

| Item | Value |
|---|---|
| Processor | Intel Core i7-2860QM CPU @ 2.50 GHz |
| GPU | nVidia GEFORCE GT 630M, 1 GB |
| RAM | 8 GB |
| Bits | 64 |

## A.4 Operating station

The operating station is a Clevo W370ET running Kubuntu 16.10. Its specifications are in Table A.4.

Table A.4: The specifications of the Clevo W370ET computer used as the operating station.

| Item | Value |
|---|---|
| Processor | Intel Core i7-3630QM CPU @ 2.40 GHz |
| GPU | nVidia GEFORCE GT 660M, 2 GB |
| RAM | 8 GB |
| Bits | 64 |

## A.5 Docker environment

For the Docker container that SPIRIT runs in, the base is from `ros:kinetic-robot`.

The following ROS packages are installed:

- ardrone-autonomy
- image-proc
- mocap_optitrack
- usb_cam

Python 2.7.13 is installed, with the following dependencies needed for running SPIRIT.

- catkin_pkg
- defusedxml
- lxml
- numpy
- pygame
- pyyaml
- rospkg
- tqdm

## A.6 External interfaces

The controller used was an off-the-shelf PlayStation3 controller.

The webcam used to record the experiments was a BSW20KM11 by iBuffalo. It is shown in Figure A.3, and its technical specifications are in Table A.5.


Figure A.3: The BSW20KM11 webcam.

Table A.5: The specifications of the iBuffalo BSW20KM11 webcam.

| Item | Value |
|---|---|
| Resolution | 1080 p |
| Framerate | 30 fps |
| Field of view | 120° |
| Microphone | Stereo |

# Part VI

# Glossaries and References

# Glossary

| | | |
|---|---|---|
| BEST | Bayesian estimation for two groups which provides complete distribution of credible values for the effect size, group means and their difference, standard deviations and their difference, and the normality of the data. By John K. Kurschke. | |
| CI | An interval such that the probability of a parameter lying in the interval is a certain value. | |
| HPD | The narrowest interval among all of the Bayesian credible intervals. | |
| ROS | An open-source collection of software frameworks for robot software development. | |
| SLAM | Constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. | |
| SPIRIT | A method which enables operators to see a model of their robot with respect to its environment, in 3D. | |
| 2D | Two dimensional space. | 3, 24 |
| 3D | Three dimensional space. | 3, 5, 7, 21 |
| AR.Drone | A remote controlled flying quadcopter built by the French company Parrot. | 3, 11, 13, 14, 23–25 |
| Docker | An open-source software project which automates the deployment of Linux applications inside software containers. | 13, 45 |
| navigation light | Also known as a running light; coloured source of illumination on various vehicles which signal the craft's heading and status. | 25 |
| OpenGL | A cross-language, cross-platform Application Programming Interface (API) for rendering 2D and 3D vector graphics. | 13, 24 |

| | | |
|---|---|---|
| Pygame | A cross-platform set of Python modules designed for writing video games. | 23, 24, 41 |
| PyMC3 | An open-source probabilistic programming framework for Python. | 30 |
| PyOpenGL | A Python wrapper for OpenGL. | 23, 24 |
| PyQt5 | A set of Python bindings for Qt. | 23, 24, 41 |
| Python | A high level general purpose computing language. | 13, 30, 45 |
| rviz | A 3D visualization tool for ROS. | 23 |

# Acronyms

| | | |
|---|---|---|
| AI | artificial intelligence | 2, 14 |
| BEST | Bayesian Estimation Supersedes the *t* Test | 30 |
| CG | computer graphics | 29 |
| CI | Credible Interval | 31, 40 |
| DOF | degree of freedom | 3 |
| DOS | degradation of signal | 3 |
| E | Effort | 36, 37 |
| EKF | Extended Kalman Filter | 8 |
| F | Frustration | 36, 37 |
| FOV | field of view | ii, 6, 7, 40 |
| FPV | first person view | 3, 4, 10 |
| GPS | Global Positioning System | 8, 18 |
| HPD | Highest Posterior Density interval | 31, 33, 34 |
| IMU | inertial measurement unit | 3, 18 |
| IP | Internet Protocol | 13 |
| JPL | Jet Propulsion Laboratory | 16 |
| LOS | loss of signal | 12, 14 |
| LRF | laser rangefinder | 5 |
| LSD-SLAM | Large-Scale Direct Monocular SLAM | 8, *see* SLAM |
| MD | Mental Demand | 36, 37 |
| mocap | motion capture | 18 |
| NASA | National Aeronautics and Space Agency | 29 |

| | | |
|---|---|---|
| NASA-TLX | NASA Task Load Index | 29, 30, 36, 37 |
| OA | Orientation awareness | 37, 38 |
| OC | Orientation control | 37, 38 |
| P | Performance | 36, 37 |
| PA | Absolute positional awareness | 37, 38 |
| PC | Absolute positional control | 37, 38 |
| PD | Physical Demand | 36, 37 |
| PM | probability mass | 31, 32, 34, 35 |
| PS3 | PlayStation 3 | 14 |
| PTAM | Parallel Tracking and Mapping | 8 |
| RA | Relative positional awareness | 37, 38 |
| RC | Relative positional control | 37, 38 |
| RMSE | Root Mean Square Error | 34, 35 |
| ROS | Robot Operating System | 10, 13–15, 18, 23, 28, 29 |
| RT-SLAM | Real Time SLAM | 8, *see* SLAM |
| SLAM | Simultaneous Localization and Mapping | 8 |
| SPIR | Teleoperation System Using Past Image Records | 5–7, 10, 41 |
| SPIRIT | Subimposed Past Image Records Implemented for Teleoperation | ii, 11, 19, 21, 25, 27–29, 31–35, 37, 38, 40, 41, 45 |
| SS | Survey Score | 37, 38 |
| TD | Temporal Demand | 36, 37 |
| TLX | Task Load Index | 36 |
| UAV | unmanned aerial vehicle | 2–5 |
| WS | Weighted TLX Score | 36, 37 |
| YAML | Yet Another Markup Language | 13 |

# Symbols

| | | | |
|---|---|---|---|
| $a$ | The $a$-component of $\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$ | $q_w$ | 16, 17, 24 |
| $b$ | The $b$-component of $\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$ | $q_x$ | 16, 17, 24 |
| $c$ | The $c$-component of $\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$ | $q_y$ | 16, 17, 24 |
| $d$ | The $d$-component of $\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$ | $q_z$ | 16, 17, 24 |
| $p$-value | The probability that, when the null hypothesis is true, the statistical summary would be the same as or more extreme than the actual observed results. | $p$ | 37, 38 |
| $x$-position | The $x$-component of the position | $x$ | 16, 17, 21, 24, 29, 31–35, 41 |
| $y$-position | The $y$-component of the position | $y$ | 16, 17, 24, 29, 31–35, 41 |
| $z$-position | The $z$-component of the position | $z$ | 16, 17, 21, 24, 41 |
| coefficient | The coefficient of an element in an evaluation function | $k$ | 21, 22 |
| current RF | The reference frame of the currently displayed frame | $C$ | 17, 21 |
| drone RF | The reference frame of the drone | $D$ | 17, 24 |
| effect size | Hedges's $g$ is an effect size used to indicate the standardised difference between two means. | $g$ | 30–32, 34–38, 40 |
| element | An element in an evaluation function | $f$ | 21, 22 |
| element index | The index of an element in an evaluation function | $i$ | 21 |
| evaluation function | An evaluation function, where $E = \Sigma k_i f_i$ | $E$ | 21 |

| | | | |
|---|---|---|---|
| global RF | The reference frame of the global coordinate system | $G$ | 17 |
| imaginary | The imaginary components of a quaternion | $\mathbf{I}$ | 17 |
| mean | The value obtained by dividing the sum of several quantities by their number; an average. | $\mu$ | 30, 32–35, 37, 38, 40 |
| number of elements | The number of elements in an evaluation function | $n$ | 21 |
| pitch angle | Rotation around the pitch axis | $\theta$ | 16, 17 |
| pitch axis | Axis perpendicular to the roll axis, in the plane of symmetry of the aircraft, positive below the aircraft | $z_b$ | 16 |
| population size | The total population size. | $N$ | 30 |
| position | A position vector in 3D | $\mathbf{x}$ | 16, 17, 20, 21, 24 |
| processing RF | The reference frame of the frame being processed | $F$ | 17, 21, 24 |
| quaternion | A quaternion in 3D | $\mathbf{q}$ | 16, 17 |
| roll angle | Rotation around the roll axis | $\phi$ | 16, 17 |
| roll axis | Axis positive out of nose of the aircraft in the plane of symmetry of the aircraft | $x_b$ | 16 |
| rotation angle | The angle by which an axis in the axis-angle representation is rotated | $\theta$ | 24 |
| rotation matrix | A rotation matrix | $\mathbf{R}$ | 17 |
| rotation vector | The vector around which the rotation happens in the axis-angle representation, in 3D | $\mathbf{e}$ | 24 |
| sample size | The size of the sample. | $n$ | 30 |
| standard deviation | A quantity expressing by how much the members of a group differ from the mean value for the group. | $\sigma$ | 30, 32–35, 37, 38 |
| state vector | A pose state vector containing the position and orientation | $\mathbf{a}$ | 17 |
| yaw angle | Rotation around the yaw axis | $\psi$ | 16, 17, 21 |
| yaw axis | Axis perpendicular to the $x_b z_b$-plane, positive out the right wing | $y_b$ | 16 |

# References


[1] H. McDaid et al. *Remote Piloted Aerial Vehicles*. Monash University. 2003-02. URL: http://www.ctie.monash.edu/hargrave/rpav_home.html (visited on 2016-08-23).

[2] C. Tetrault. *A Short History of Unmanned Aerial Vehicles (UAVs)*. Draganfly. 2009-03. URL: http://www.draganfly.com/blog/a-short-history-of-unmanned-aerial-vehicles-uavs/ (visited on 2016-08-23).

[3] *WALKERA DEVO F12E FPV 5.8G Transmitter Receiver Telemetry Radio Controller 32CH*. Ebay. 2016-12. URL: http://www.ebay.com/itm/WALKERA-DEVO-F12E-FPV-5-8G-Transmitter-Receiver-Telemetry-Radio-Controller-32CH-/141432435221 (visited on 2016-12-16).

[4] *Phantom 3 Standard - Remote Controller*. DJI. 2016-06. URL: http://www.dji.com/phantom-3-standard/remote-controller (visited on 2016-06-20).

[5] *27 Awesome FPV Goggles and Gadgets - Killer FPV Drone Systems!* Dronethusiast. 2015-07. URL: http://www.dronethusiast.com/the-ultimate-fpv-system-guide/ (visited on 2016-06-20).

[6] *AR Drone-side booster and antenna in numbers. Tests.* drone-forum.com. 2013-07. URL: https://www.drone-forum.com/forum/viewtopic.php?t=6712 (visited on 2016-06-22).

[7] J. Serna. *Drone knocks out power to hundreds of West Hollywood residents*. Los Angeles Times. 2015-10. URL: http://www.latimes.com/local/lanow/la-me-ln-drone-power-west-hollywood-20151027-story.html (visited on 2015-12-11).

[8] *Woman knocked unconscious by falling drone during Seattle's Pride parade*. The Seattle Times. 2015-06. URL: http://www.seattletimes.com/seattle-news/crime/woman-knocked-unconscious-by-falling-drone-during-seattles-pride-parade/ (visited on 2015-12-12).

[9] *Toddler's eyeball sliced in half by drone propeller*. BBC News. 2015-11. URL: http://www.bbc.com/news/uk-england-hereford-worcester-34936739 (visited on 2015-12-21).

[10] J. Elliott. *Drone flight near Vancouver airport attracts Transport Canada, RCMP attention*. CTV News. 2014-04. URL: http://www.ctvnews.ca/canada/1.1788494 (visited on 2015-12-11).

[11] J. T. Hing, K. W. Sevcik, and P. Y. Oh. "Development and evaluation of a chase view for UAV operations in cluttered environments". In: *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*. Springer. 2009, pp. 485–503.

[12] C. W. Nielsen, M. A. Goodrich, and R. W. Ricks. "Ecological interfaces for improving mobile robot teleoperation". In: *IEEE Transactions on Robotics* 23.5 (2007), pp. 927–941.

[13] A. Kelly et al. "Real-time photorealistic virtualized reality interface for remote mobile robot control". In: *The International Journal of Robotics Research* 30.3 (2011), pp. 384–404.

[14] N. Shiroma et al. "A Novel Teleoperation Method for a Mobile Robot Using Real Image Data Records". In: *Robotics and Biomimetics, 2004 IEEE International Conference on*. IEEE. 2004, pp. 233–238.

[15] M. Sugimoto et al. "Time Follower's Vision: a teleoperation interface with past images". In: *IEEE Computer Graphics and Applications* 25.1 (2005), pp. 54–63.

[16] M. Ito et al. "A Teleoperation Interface using Past Images for Outdoor Environment". In: *SICE Annual Conference*. 2008, pp. 3372–3375.

[17] R. Murata et al. "Teleoperation system using past image records for mobile manipulator". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE. 2014, pp. 4340–4345.

[18] *Rotation Matrix*. Wikipedia. 2016-06. URL: `https://en.wikipedia.org/wiki/Rotation_matrix#Quaternion` (visited on 2016-06-22).

[19] J.-L. Blanco. *A tutorial on SE(3) transformation parameterizations and on-manifold optimization*. Tech. rep. #012010. Machine Perception and Intelligent Robotics Research Group, University of Málaga, 2013-05, p. 14. (Visited on 2017-02-25).

[20] J. K. Kruschke. "Bayesian estimation supersedes the t test." In: *Journal of Experimental Psychology: General* 142.2 (2013), p. 573.

[21] S. S. Sawilowsky. "New effect size rules of thumb". In: (2009).

[22] S. G. Hart. "NASA-task load index (NASA-TLX); 20 years later". In: *Proceedings of the human factors and ergonomics society annual meeting*. Vol. 50. 9. Sage Publications Sage CA: Los Angeles, CA. 2006, pp. 904–908.