Delft University of Technology

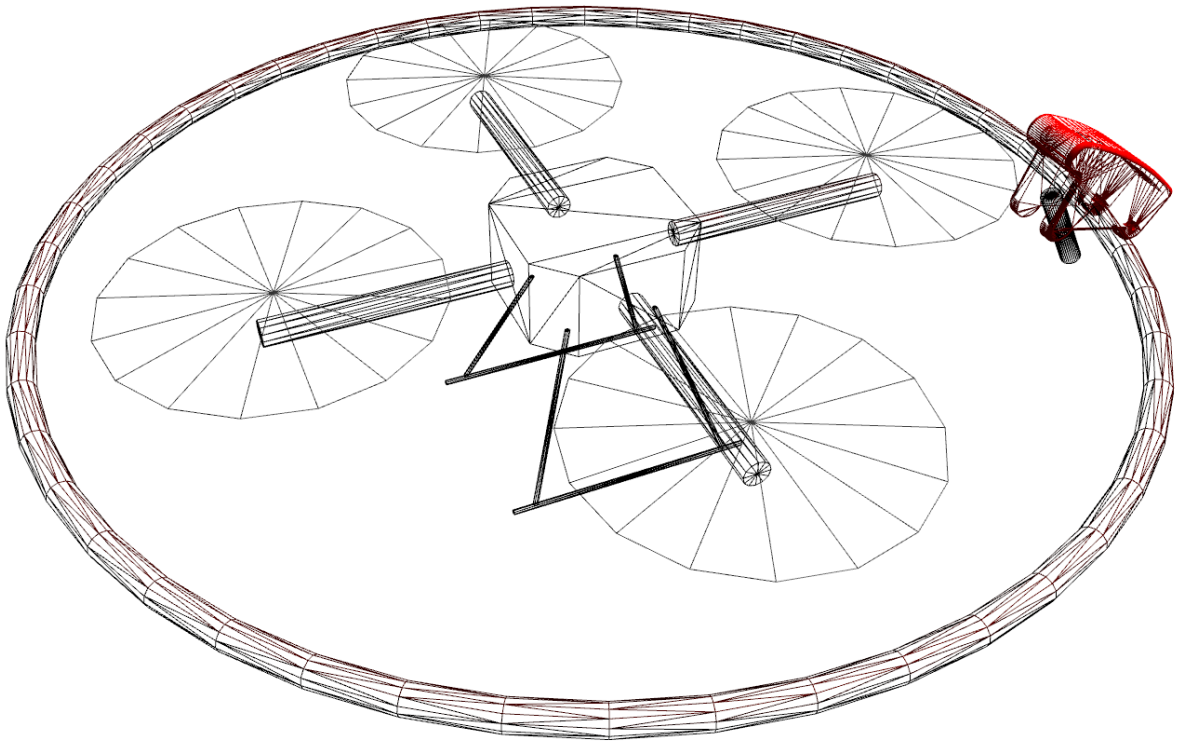
Effective Human-Machine Interfaces for Aerial Telemanipulation

Master thesis

Author:
Graham Nixon

Supervisors:
Ir. Jeroen Wildenbeest
Dr. Ir. David Abbink

December 14, 2015



Preface

In the process of becoming a mechanical engineer, I was taught a great deal about mechanics, calculus, algebra and programming. But it was not until my master degree that things really started to become interesting for me and I believe this was because a new element was added to the equation: the Human. A great deal of mechanical devices require some form of physical interaction with humans such as cars, planes and robotic devices and it is this subject that not only interests me but that was my focus for this research project.

Today, drones or Unmanned Aerial Vehicles (UAVs) are a hot topic and they seem to be a significant part of technological news every day. Besides nice videos of aerial views, we are well on our way to being able to use UAVs to physically perform tasks which is a compelling addition to their capabilities because of their freedom. A well researched topic is how to create an effective interface to perform tasks with a robot in a remote environment but nothing is known as to how this compares to performing tasks with a UAV nor how such an interface should look.

The goal of this study is to obtain a better understanding of how a human operator can best control an Unmanned Aerial Vehicle to perform physical tasks in a remote environment. This is described extensively in a scientific paper which we aim to publish. Furthermore, this thesis consists of appendices which provide a comprehensive overview of the accomplishments during this study. The main objective of these appendices is to give the reader insight into my "train of thought", provide background information and additionally give sufficient insight for future researchers to understand and optionally perform follow up studies.

I would primarily like to thank my supervisors: Jeroen and David for the brainstorms, time, effort, and support during the last year. Additionally I would like to thank Jeff van Egmond and Machiel Bruinink for helping me with my software issues when no one else could. And last but not least, I would like to thank everyone from our study group in the BME/BMD tower on the fourth floor. Without you guys, I would not only have been less motivated but my graduation project would have also simply been a lot less fun.

- Graham Nixon
December 14th, 2015

Contents

1 Scientific Paper **6**

A Quick start guide **16**
A.1 Installation 16
A.2 Run 19

B Software implementation **25**
B.1 Task design 26
B.2 Controller design 28
B.3 Haptic feedback design 30
B.4 Gazebo plugins 32
B.5 ROS nodes 35
B.6 Data acquisition 36

C Experimental setup and protocol **38**
C.1 Setup 38
C.2 Protocol 39
C.3 Instructions 41
C.4 Informed consent 42

D Measuring effectiveness **44**
D.1 Objective metrics 44
D.2 Subjective metrics 53

E Pilot results **56**
E.1 Results 56
E.2 Changes 60

F Experimental results **63**
F.1 Participants 63
F.2 Results 63

Bibliography **70**

List of Figures

A.1 Graphical User Interface (GUI) to start the simulation . . . 23

B.1 A schematic overview of how the simulation implementation works . . . 25
B.2 Task taxonomy including potential disaster tasks and UAV performed tasks . . . 26
B.3 Three realistic and realistic disaster tasks . . . 26
B.4 Safety switch virtual model . . . 27
B.5 Dirty wall virtual model . . . 28
B.6 A first look at Gazebo contact forces . . . 31
B.7 Raw and rate limited Gazebo forces . . . 31
B.8 Comparing the effect of a rate limiter to that of a 2nd order filter . . . 32
B.9 The two status possibilities for the safety switch model . . . 33
B.10 Gazebo GUI widget showing task completion . . . 34
B.11 How to control the UAV with a keyboard using the keyboard control node . . . 35

C.1 Experimental Setup with all the components . . . 39
C.2 Experimental conditions and protocol . . . 40
C.3 Experimental setup training . . . 41
C.4 Randomized trials . . . 41
C.5 Instruction video . . . 42

D.1 Representative trials showing trajectory length metric . . . 46
D.2 Simple and more elaborate reversal count . . . 48
D.3 Representative trials showing reversal rate metric . . . 49
D.4 Representative trials showing average contact force metric . . . 50
D.5 Representative trials showing the contact transitions metric . . . 52
D.6 Van Der Laan questionnaire items . . . 53
D.7 Page 1 of questionnaire: Personal questions . . . 54
D.8 Page 2 of questionnaire: Interface acceptance . . . 55

E.1 Legend showing the markers used for each figure in this chapter . . . 56
E.2 Pilot results: Completion time metric . . . 56
E.3 Pilot results: Trajectory distance metric . . . 57
E.4 Pilot results: Cumulative number of flips metric . . . 57
E.5 Pilot results: Standard deviation of the input metric . . . 58
E.6 Pilot results: Number of reversals metric . . . 58
E.7 Pilot results: Average contact force metric . . . 59
E.8 Pilot results: Maximum contact force metric . . . 59
E.9 Pilot results: Maximum velocity metric . . . 60
E.10 Training world for (T_0) . . . 61

F.1 Experimental results: Completion time metric . . . 64
F.2 Experimental results: Trajectory distance metric . . . 64
F.3 Experimental results: Cumulative number of flips metric . . . 65
F.4 Experimental results: Standard deviation of the input metric . . . 65
F.5 Experimental results: Number of reversals metric . . . 66
F.6 Experimental results: Average contact force metric . . . 66
F.7 Experimental results: Maximum contact force metric . . . 67

F.8 Experimental results: Maximum velocity metric 67
F.9 Experimental results: Number of contact transitions metric 68
F.10 Experimental results: Inefficient time metric 68

List of Tables

A.1 Argument list for ROS launch file 20
A.2 World list options for ROS launch file 21
A.3 Controller types for ROS launch file 21

F.1 Subject data from experiment 63
F.2 Raw subject data from acceptance questionnaire 69

Effective Human-Machine Interfaces for Aerial Telemanipulation

Graham A.N. Nixon, Jeroen G.W. Wildenbeest and David A. Abbink

Abstract—Most ground robots deployed in disaster areas have been of limited use mainly because of their reduced mobility in degraded environments. Unmanned Aerial Vehicles (UAVs) have considerable mobility benefits over ground robots, however they are currently only used for providing aerial views and often lack any manipulation capabilities, despite their proven ability to perform small force tasks. Research in ground robots has repeatedly stated that there is need for better human-machine interaction, while interface design for aerial robots is an unexplored area where generally primitive interfaces are used. The objective of this study is to understand what type of interface design is most beneficial for telemanipulation with a UAV. In a human factors experiment, subjects (N = 14) were instructed to open and flip a safety switch in a virtual reality environment using a bumper surrounding the UAV and to do so as fast and careful as possible. Five different interface designs were compared in this study. First, a 3 degrees-of-freedom (DoF) manipulator with position control was compared to the typical baseline interface: a gamepad with rate control. Second, the effect of having haptic feedback on the manipulator was tested. And finally performance on both the manipulator, with and without haptic feedback, was compared to having a fully actuated, grounded slave, with and without haptic feedback. Results show that a 3-DoF manipulator with position control substantially outperforms a gamepad with rate control. Haptic feedback had no significant influence on the task execution of the operator. Using a grounded slave was significantly more effective than using an aerial slave though there was no relative effect of haptic feedback between the grounded and the aerial slave. This study has shown that interface design can have a significant effect on performing remote manipulation tasks with UAVs.

Index Terms—Aerial, telemanipulation, tele-manipulation, interface, UAV, haptics.

1 Introduction

Disaster areas are very difficult to operate in: they are too dangerous or impossible to access for humans. A recent example of such a disaster is that of the Daiichi Fukushima power plant in Japan. This type of accident, though rare, illustrates the need for unmanned vehicles to be deployed into disaster areas to monitor the area and perform critical tasks.

Robots were very first deployed in a search and rescue mission just after the September 11 attacks on the Twin Towers in the United States [1]. Since then robots have been used in at least 43 different disaster scenarios when human urban search and rescue was too risky [2], [3]. Additionally a great number of organisations and competitions are focussing on creating robotic solutions to aid humans in the event of disasters [4], [3], [5], [6], [7].

Despite these efforts and the imaginings of science fiction authors, the current state-of-practice of robots designed and deployed in disaster areas has significant limitations. One inherent limitation of the majority of these robots is that they have difficulty manoeuvring in degraded and unknown environments such as disaster areas where the ground is generally not level and often unstable [8], [1].

- *G.A.N. Nixon is with the Delft Haptics Lab, Deparment of BioMechanical Engineering , Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands.*
- *J.G.W. Wildenbeest is with the Delft Haptics lab, Department of BioMechanical Engineering, Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands and with the Heemskerk Innovative Technology B.V., Jonckerweg 12, 2201 DZ Noordwijk, The Netherlands.*

Aerial robots have considerable mobility benefits over ground robots but have previously only been used for acquiring imagery data from disaster areas. Notwithstanding the fact that Unmanned Aerial Vehicles (UAVs) lend themselves perfectly for aerial views, the ability to actually perform physical tasks is also important in disaster sites [9]. Expanding UAV capabilities to be able to carry and manipulate objects they encounter, is interesting because of their unique abilities to position themselves anywhere in space, regardless of the terrain [10], [11].

There is a large variety of potential manipulation tasks in disaster areas which can be generally classified as either tool assisted tasks or fine motor control tasks. There are light weight tool assisted tasks (e.g. cleaning a surface) or more demanding force tasks (e.g. breaking down a wall). Also there are light weight fine motor control tasks (e.g. flipping a switch) or more insistent force tasks (e.g. turning a valve). It is in the very small force regions where manipulation has been demonstrated with UAVs. Lifting and gripping with vehicle-to-payload ratios of up to $m_{\text{uav}}/m_{\text{lift}} \approx 1$ has been achieved [12], [13]. Slightly more intricate force tasks like turning a valve and interacting with humans have been achieved with interaction forces on the order of $5\,\text{N}$ [14], [15], [16], [11]. More dexterous manipulation tasks have been achieved by adding multiple robotic manipulators to UAVs for 'peg-in-hole' tasks or contact for non destructive testing [17], [18], [19]. Efforts are also being made in practice, where organisations and competitions are focussing on environment manipulation such as: Airobots [20], Aerial Robotics Competition [21], Aerial Robotics Cooperative Assembly System [22] and Aeroworks [23]. All these efforts proove that UAVs are highly capable of performing manipu-

lation tasks, as long as the nessasary forces remain relatively low.

A fundamental difference between the research that has been done in aerial manipulation and the requirements for robots in disaster areas is that the latter often require that tasks be performed manually, on-the-fly, with the human-in-the-loop. "The irony of semiautonomous (and to some extent autonomous) systems is that teleoperations will be necessary under the most difficult situations when the underlying robotic intelligence and sensor capabilities require human intervention" [24]. This is why the human will provisionally play a vital part in manually controlling robots to do physical work in such environments.

We believe there are a number of factors hindering UAVs from being used for performing remote manipulation tasks in disaster areas. Firstly, there are still numerous engineering challenges to overcome, where not only the complexity and variability of disaster areas poses problems, but also the endurance and power of current UAVs limit their capabilities. Secondly, manipulation tasks, which are generally designed specifically for humans, are even harder for robots to perform, especially in remote environments and with limited perceptual information. Though telemanipulation with grounded vehicles is a relatively well-researched subject, very little is currently known with regards to telemanipulation utilizing an aerial slave.

In both grounded telemanipulation research as well as surveys of robots deployed in disaster areas, an often emphasized and important theme is that of the human-machine interfacing [25]. In robot-aided disaster research, it has been stated repeatedly that there is a need for better human-robot interaction [2]. Creating effective and more intuitive interfaces is therefore of great importance. In research and also in practice, UAVs are most often operated with simple gamepad-like interfaces that control the velocity of the UAV. This would seem beneficial because the workspace of UAVs is often very large or indefinite. However when the workspace is small (e.g. doing household manipulation tasks), it is known that position control allows for more precise and accurate tracking [26]. Thus for executing tasks remotely, a position control manipulator is hypothesized to be superior when compared to the current industry standard— a legacy gamepad or radio-controlled remote with rate control.

"Situation awareness is the major bottle neck in robot autonomy, not autonomous navigation. Results from field studies with 33 operators show that the robot is stationary half the time, as operators try to understand what is going on around them by communicating about the task, system and environment" [27]. In general, studies in grounded telemanipulation report positively on providing operators with haptic feedback [28], [26], [29], [30]. It is therefore hypothesized that haptic feedback is also beneficial during aerial telemanipulation, just like with grounded telemanipulation. Because of the complex dynamics of a UAV compared to a grounded slave, we also expect the effect of haptic feedback to be larger in aerial telemanipulation. All these factors likely influence task execution in some way although it is not known how, nor is it known how a control interface should be designed for effective telemanipulation with a UAV.

The goal of this research is: to understand the fundamentals as to how interface design influences task execution during aerial telemanipulation. Various factors will be studied to determine how they influence task execution during aerial telemanipulation, namely: the master device, the master device's controller, haptic feedback and the slave dynamics.

Based on the goal stated above, it is hypothesized that:

H_1: A position controlled, 3 degrees-of-freedom (DoF) manipulator interface increases performance over a conventional interface in rate control

H_2: Haptic feedback increases performance over not having haptic feedback

H_3: Higher performance is achieved with a grounded slave compared to on an aerial slave. In addition, the relative effects of haptic feedback are larger with an aerial slave.

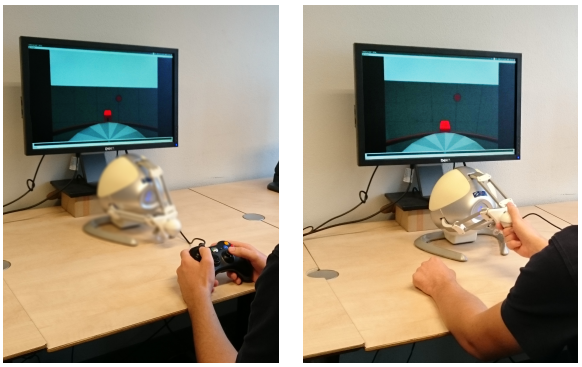
2 METHOD

2.1 Subjects

14 subjects (11 male and 3 female) ranging in age from 23 to 36 years (M: 25.2 years and SD: 3.3 years) participated in this human-factors experiment. Two subjects had previous experience piloting UAVs and five had (self-evaluated) substantial console gaming experience. The subjects were shown a short introduction video where the experiment and the interface designs were introduced. All subjects gave their informed consent before participating in the experiment. Furthermore, the experiment was approved by the Delft University of Technology Human Research Ethics Committee (HREC).

2.2 Hardware

The experimental setup consisteed of two different master devices (interfaces), a laptop (running Ubuntu 14.04, ROS Indigo and Gazebo 5.1) and a secondary screen placed in front of the interfaces (Fig. 1). The interfaces used were a gamepad (an XBox controller, Fig. 1a) and a 3-degree-of-freedom manipulator (Novint Falcon, Fig. 1b). The gamepad had a deadband of 10 % to prevent undesired drift.



(a) Gamepad (b) 3-DoF manipulator

Fig. 1: Subject performing a trial with the two master devices that were used during this experiment

2.3 Simulation software

A simulation was used for both the slave and the environment for three reasons. Firstly, because to our knowledge, we are first to conduct such aerial telemanipulation research and no such actual platform yet exists. Secondly, because the focus of this research is on learning what factors influence the effectiveness of an interface for aerial telemanipulation and thus the slave side of the telemanipulation is of less concern. Other research has focussed on optimizing the robotic slave for UAVs which will likely further improve task performance [11], [17], [18], [19]. The third reason is that the modular design approach paves the way for future experiments where physical validation could be proven with real UAVs.

The base of the simulation consisted of a package called `hector_quadrotor`, running at 1000 Hz [31]. The choice of this software was primarily due to it being highly comprehensive, customizable and the fact that it has been validated with real flight data [32]. This simulation was based on the Robotic Operating System (ROS) as a framework and Gazebo as physics and visuals simulator. The simulation included optional modules for motor and propeller dynamics, aerodynamics, external disturbances (e.g. wind), noisy sensor signals and state estimation. For this experiment additional ROS nodes and Gazebo plugins were written to complement the simulation to together form a aerial telemanipulation research platform. Yaw input (rotation around the z-axis) was eliminated to simplify the experiment.

The UAV model used in the simulation was a quadcopter which is a non-holonomic vehicle. Quadcopters are often used for research because of their manoeuvrability and ability to carry small payloads [33]. The working of such a UAV is similar to that of a helicopter where a body contains four rapidly spinning rotor blades which push the air downwards, creating a thrust force keeping the vehicle aloft. Variation of the rotor speeds allows for instantaneous movement in z-plane and all rotations. Movement in the horizontal plane requires tilting the quadcopter by selective variation of individual rotor thrust. The model used in the simulation has the specifications as defined below (Fig. 2).

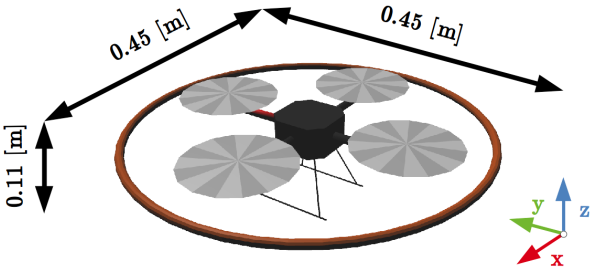
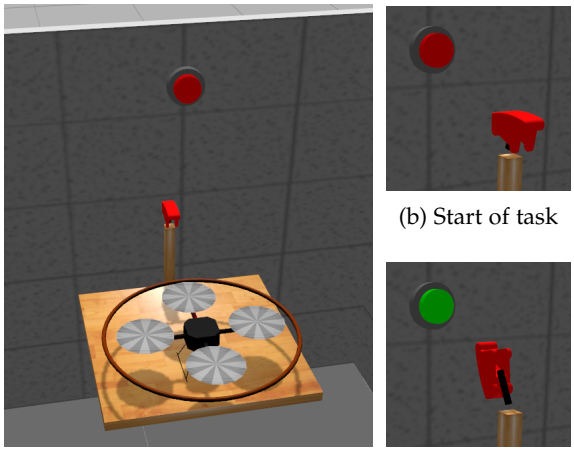


Fig. 2: The virtual UAV that was used to perform the task. The orange bumper was used to exert the forces to perform the task. The axis definition is also shown in the graph with arrows. This UAV has the following specifications: $m = 1.477\,\text{kg}$, $I_{xx} = I_{yy} = 0.1152\,\text{kgm}^2$, $I_{zz} = 0.0218\,\text{kgm}^2$.

The UAV model was scaled to half the original size to have a better relationship with the relatively small task that had to be performed. Other dimensional parameters such as rotor diameter were collectively scaled down but mass and inertia were held the same as these were still in the correct order of magnitude compared to commercial UAVs.

2.4 Task Description

A representative manipulation task was chosen to guarantee scalability of this research. Simple gripping tasks have already been accomplished and large force tasks pose engineering difficulties. The chosen task was to activate a safety switch (Fig. 3a). To flip the switch, the cover of the safety switch has to first be opened (Fig. 3b), after which the switch under the cover can be flipped (Fig. 3c). The cover and switch were set to have a mass of respectively 20 g and 10 g. The joint dynamics of the cover, which created the most difficulty of the task, were set to have a damping of $0.05\,\text{N m s rad}^{-1}$ and a static friction of $0.02\,\text{N m}$. The switch had the same damping but half the static friction, $0.01\,\text{N m}$. These values were chosen experimentally to make the task difficult, but if the UAV was positioned correctly, not impossible. To emulate a simple slave, a torus shaped bumper surrounding the UAV was used to exert forces on the environment. Commercial UAVs are often equipped with such bumpers to protect the rotors from damage. In this experiment the UAV was initialliy positioned in close vicinity to this task, and thus the approach phase, an important but inherently different problem, was not researched here.



(a) Task overview (b) Start of task (c) Task complete

Fig. 3: Simulation view showing the task environment from a third person view (Fig. 3a), the start of the task (Fig. 3b), and the completed task (Fig. 3c). During the experiment the task was viewed from a first person camera mounted on top of the UAV.

Subjects were instructed to perform the task as quickly and as carefully as possible. A UAV in contact with the environment can be a very unstable situation where if too much force is applied, the UAV can flip over, which is obviously not desirable. Therefore a trade-off must be made between speed and caution. If the UAV was flipped during the task, the trial would have to be repeated and the number of flips was recorded. The initial position of the UAV was

on a platform about 5 cm in front of and 10 cm below the switch.

2.5 Controller and haptic feedback design

The type of controllers and the haptic feedback design highly influenced the intuitiveness and thus the performance of the subjects. The position and rate controller of the UAV were further developed from the initial cascaded PID-controllers used in the simulation [32]. As these were optimized for free flight, a number of changes were made.

The rate controller was only used for the gamepad condition (I) and was left mainly unaltered except for the integral action which was removed. This was done to prevent undesired tilting of the UAV when in contact with the environment when the reference velocity was set to zero. Also, an additional P-controller was added to control the yaw angle of the UAV. Because the rate controller only governs the reference velocities, an additional controller was needed to keep the UAV pointing forwards if the yaw angle changed due to contact with the switch. Furthermore the maximum velocity (full joystick deflection) was set to $0.2\,\mathrm{m\,s^{-1}}$. This limit was low enough to allow precise movement of the UAV and fast enough to reach the same maximum velocity as could be reached with the position controller.

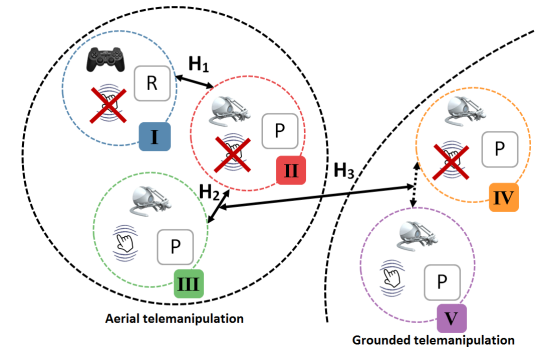
The position controller was used for the manipulator and the manipulator with haptic feedback condition (II and III). The workspace of the 3-DoF manipulator is approximately $10 \times 10 \times 10$ cm though the actual workspace is significantly less due to its delta-robot form. To give the subjects enough room to position the UAV around the switch in it's open position and not come near the end stops of the workspace, the manipulator to workspace scaling was set 1:4.

For the design of the haptic feedback, the simulated contact forces had to be reshaped due to the very high initial contact forces in Gazebo. In attempt to smooth these forces before applying them to the manipulator, a rate limiter was implemented which limited the maximum force change per step to be: 0.2 N— which is approximately 10 % of the maximum force which would occur during manipulation. Such a rate limiter has a comparable effect to a 110 Hz low pass filter and is thus justified because it is far above the human force control bandwidth.

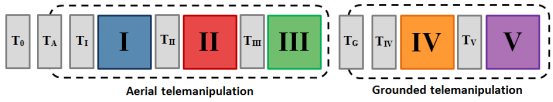
For the grounded telemanipulation conditions (IV and V), a different position controller was implemented. Here the slave dynamics were changed such that the UAV was fully actuated, acting like a second order system floating in space with stiffness and damping as a result of the PD-controller. The (critically dampened) PD-controller was introduced to track the desired position. Furthermore the force/torque and velocity limits were deactivated to simulate a stronger slave.

2.6 Experimental Design

In this experiment three independent variables were altered: the interface type (gamepad or 3-DoF manipulator with their respective controllers), the presence of haptic feedback, and the UAV slave dynamics (grounded or aerial) (Fig. 4a). The comparison with the baseline approach, the gamepad with rate control, was only compared to the 3-DoF manipulator with position control. The presence of haptic feedback and the type of slave dynamics was compared overall which resulted in five experimental conditions.



(a) Experimental conditions)



(b) Experimental protocol

Fig. 4: The experimental conditions shown in relation to other literature (Figure 4a) and as experimental protocol (Fig. 4b). The five experimental conditions indicated as I, II, III, IV and V with their respective color which are used throughout this paper. The three hypothesis (H_1, H_2 and H_3) and their respective relationships are shown as arrows. The icons represent the interface type (gamepad or manipulator), whether haptic feedback is present, and the type of controller: rate (R) or position (P).

Each experimental conditions was preceded by training for that specific condition (Fig. 4b). These training conditions were repeated a minimum of two times and continued until the subject had performed the task successfully at least two times. Each experimental condition was followed up by a questionnaire to measure the subject's subjective acceptance of the interface design. Furthermore each experimental condition was repeated 5 times, to reduce within-subject variance.

These five experimental conditions were grouped by their slave dynamics type. Subjects would start with either aerial or grounded slave dynamics and then continue with the internal conditions of these groups, which were also randomized.

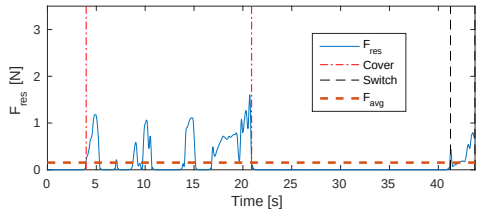
2.7 Data acquisition

To generate metrics of each trial, the following data streams were recorded from the ROS framework: position and orientation of the UAV, force acting on the UAV bumper, feedback force provided to the manipulator, position data of gamepad and manipulator and their respective buttons, angle of the switch cover, angle of the switch and real time factor of the simulation. The feature `rosbag` was used to record published messages from ROS to a file. All data streams were recorded at 1000 Hz except for the gamepad input signals and the state of the UAV, which were recorded at 100 Hz.

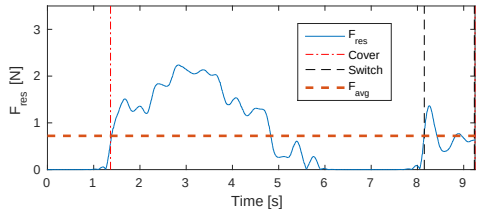
2.8 Metrics

The effectiveness of the interface design was evaluated with a number of metrics which capture a performance characteristic in a single value. To account for the speed-accuracy trade off, both performance and effort metrics were taken into account. The following metrics were used to measure of the effectiveness of the interface:
Performance:

- *Completion time* [s]: Elapsed time between the start of the task (push of an interface button) and the switch being fully activated. The subject must keep the UAV level/stable even after the switch has been flipped. Subjects are considered to have performed better with lower completion times.
- *Average contact force* [N]: The average contact force acting from the environment onto the UAV bumper. Subjects are considered to have performed better with a higher average force. Here the resultant force was used, filtered by a 3rd order 4-Hz low-pass filter, measured from the press of the interface button to the switch being fully flipped, to calculate the average force.



(a) Trial with low average contact force (low performance)

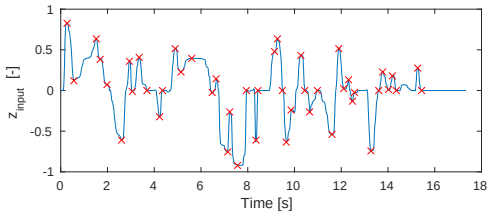


(b) Trial with high average contact force (high performance)

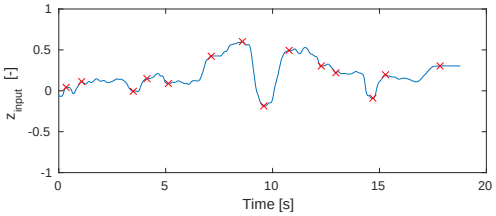
Fig. 5: Illustration of the average contact force metric for a trial with low (Fig. 5a) and high (Fig. 5b) average contact force.

Effort:

- *Reversal rate* [-]: The number of steering corrections on the interface (Fig. 6), filtered at 4-Hz low-pass (3rd order). Trained subjects tend to steer smoothly which results in a low number of reversals. The reversals of the x-axis were not taken into account because the task requires only manipulation in the vertical (y and x-) plane. Therefore both these reversals were be analysed. The number of reversals depends on a number of factors:
  - $\frac{d(x_{\text{input},i})}{dt} = 0$, the steering input changes direction where $x_{input,i}$ is the master input input position and i is 1 or 2 for respectively y or z.
  - $c = 0.1$, the threshold value for change in input (set to 10 % of the maximum joystick deflection). The absolute change in input since the last steering correction must exceed this value in order to be counted as a reversal.



(a) Reversals from trial with gamepad interface (high effort)



(b) Reversals from trial with manipulator interface (low effort)

Fig. 6: Illustration of the reversals metric for a trial with a low (Fig. 6a) and a high (Fig. 6b) number of reversals. The reversals are marked with red crosses

2.9 Data analysis

To analyse the difference between the means of the trial sets for each experimental condition, a repeated measures ANOVA was used. For H_3 results were also grouped with factors *slave dynamics* and *haptics*. Each subject performed five trials per experimental condition to reduce inter-subject variance. The five trials per subject were averaged to give a single value per subject per experimental condition.

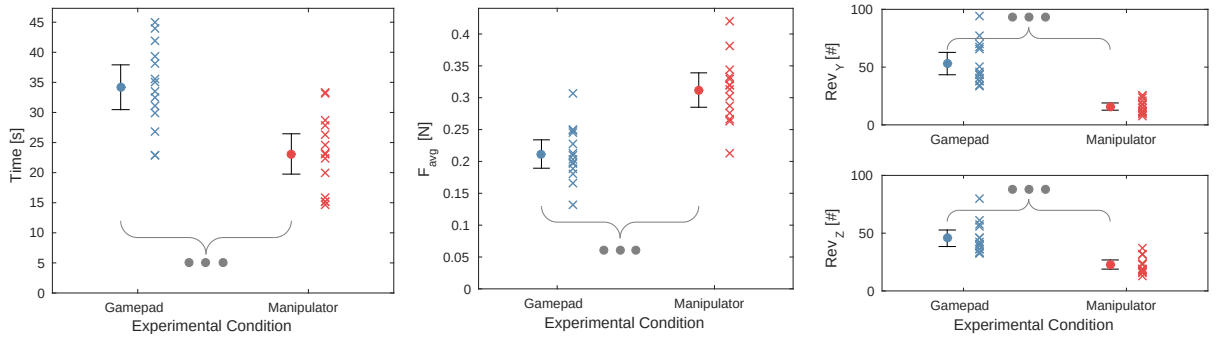
Normality assumption was checked on all metrics to ensure the applicability of the statistical tests used ($p < 0.05$). Results are regarded as statistically significant with $\alpha = 5\%$ ($p < 0.05$).

Post-hoc, because multiple comparisons are being done with the same data set, a Bonferroni correction was applied. This correction was chosen above others as it is generally suitable for testing specific hypotheses as opposed to testing all possible comparisons. Data was tested on three separate hypotheses therefore the p-values were multiplied by 3 to test their significance.

3 Results

The figures in this section show the mean and 95 % confidence interval of all subjects, averaged over their 5 repetitions. Subject averages are shown as crosses next to their respective means and confidence intervals.

All metrics (completion time, average contact force and number of reversals) are highly significant over all five experimental conditions ($p < 0.001$). Because not all comparisons are of interest here, the results were examined specifically per hypothesis. Significant results are denoted with '•', '••' and '• • •' for respectively $p < 0.05$, $p < 0.01$ and $p < 0.001$.



(a) Task completion time (b) Average contact force (c) Number of master input reversals from the y and z axis.

Fig. 7: Results from the comparison between the gamepad with rate control and the 3-DoF manipulator with position control (H_1). Figures show the average and the 95 % confidence intervals as well as the individual subjects marked (crosses).

	T [s]	F_{avg} [N]	Rev_y [-]	Rev_z [-]
I.	34.19 (7.10)	0.21 (0.04)	53.07 (18.49)	45.59 (13.65)
II.	23.11 (6.40)	0.31 (0.05)	15.84 (5.96)	22.84 (7.59)
III.	27.20 (8.09)	0.30 (0.04)	20.80 (9.67)	31.17 (15.06)
IV.	9.30 (3.43)	0.38 (0.08)	6.47 (2.90)	11.89 (3.76)
V.	9.49 (2.94)	0.39 (0.09)	7.83 (2.13)	13.9 (2.62)

TABLE 1: The mean and standard deviation (in brackets) of all the metrics of the five experimental conditions. T is the task completion time, F_{avg} is the average contact force and Rev_y and Rev_z are the number of master input reversals in y and z direction.

3.1 Effects of master device

Hypothesis 1 (H_1) deals with the resulting effects from the change of master device: a gamepad with a rate controller (as baseline) and the 3-DoF manipulator with a position controller. Fig. 7a presents the completion time of these two conditions. The change in master device and their respective controllers significantly shortens the completion time ($F(1,13) = 48.02, p < 0.001$). Fig. 7b shows the average contact force for the same two conditions. Here the average contact force is significantly higher for the 3-DoF manipulator with position control, as compared to the gamepad with rate control ($F(1,13) = 44.56, p < 0.001$). As for a measure of effort, Fig. 7c shows the number of reversals for the same two conditions. The change in interface design and their respective controllers significantly reduces the amount of reversals in the $y-$ and z-axis respectively ($F(1,13) = 61.73, p < 0.001$) and ($F(1,13) = 28.80, p < 0.001$).

3.2 Effects of haptic feedback

Hypothesis 2 (H_2) analyses the effects of of having haptic feedback or not on the manipulator. Results show that having haptic feedback does not effect the completion time of the task ($p = 0.116$). Neither does it have an effect on the average contact force or the number of reversals in y-direction ($p = 0.879$ and $p = 0.059$). It does however cause a significant increase in the number of reversals in z-direction ($F(1,13) = 8.03, p < 0.05$).

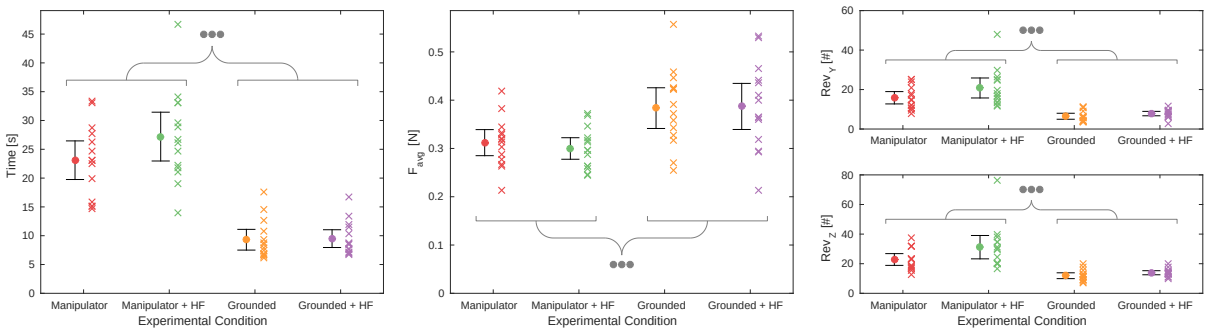
3.3 Effects of slave type

Hypothesis 3 (H_3) covers the effect of the slave type (grounded or aerial) on both conditions with and without haptics. Fig. 8a shows completion times for both the grounded and the aerial slaves, with and without haptic feedback. Completion time was affected by the slave type—the task on the grounded slave was performed significantly faster ($F(1,13) = 108.28, p < 0.001$). Also, the average force exerted (Fig. 8b) significantly increased when using a grounded slave ($F(1,13) = 18.58, p < 0.001$). The effort (measured as the number of reversals) showed significant reduction in both y and z direction ($F(1,13) = 48.99, p < 0.001$) and ($F(1,13) = 35.96, p < 0.001$) respectively (Fig. 8c). There was also a significant interaction effect from the haptic feedback on the completion time ($F(1,13) = 2.01, p < 0.05$). Furthermore the results show that there was no interaction effect of having haptic feedback, on the average contact force ($p = 1$). The subject effort (measured as the number of reversals) in both y and z direction, was also not affected by adding haptic feedback ($p = 0.159$ and $p = 0.097$ respectively).

3.4 Subjective acceptance questionnaire

As a subjective measure of the acceptance of the interfaces, usefulness and a satisfactory scores were computed using the Van Der Laan five point rating scale [34]. This scale gives results in an usefulness and a safisfaction score on the interval $[-2, 2]$. The average of all satisfaction and usefulness scores was then calculated (Fig. 9).

The manipulator was found to have a higher satisfaction score ($F(1,13) = 8.47, p < 0.05$) as well as a higher usefulness score ($F(1,13) = 9.18, p < 0.05$) compared to that of the gamepad. There were no significant differences for the usefulness score or the satisfaction score for having haptic feedback or not ($p = 0.448$ and $p = 1$ respectively).



(a) Task completion time (b) Average contact force (c) Number of master input reversals from the y and z axis.

Fig. 8: Results from comparison of the 3-DoF manipulator, with and without haptic feedback with aerial and grounded slave dynamics (H3). Figures show the average and the 95 % confidence intervals as well as the individual subjects (crosses).

	Usefulness [-2,2]	Satisfaction [-2,2]
I.	-0.19 (0.80)	-0.63 (1.07)
II.	0.43 (0.61)	0.34 (1.02)
III.	0.74 (0.78)	0.43 (1.02)
IV.	0.99 (0.60)	1.24 (0.48)
V.	1.14 (0.37)	1.27 (0.70)

TABLE 2: The mean and standard deviation (in brackets) of the usefulness and satisfaction scores of the five experimental conditions based on the 5 point rating scale developed by Van der Laan [34]

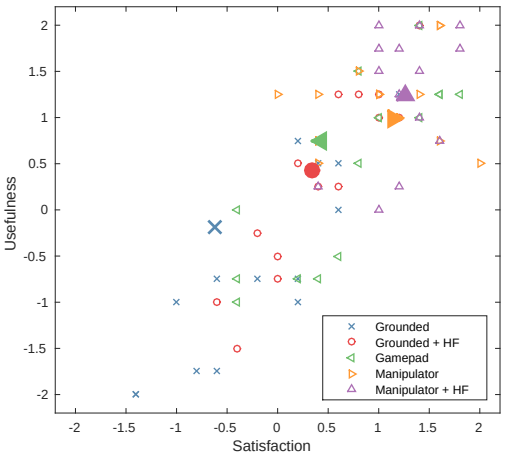


Fig. 9: Results of the subjective acceptance questionnaire with averaged (large solid markers) and subject individual (small hollow markers) satisfaction and usefulness scores of the five interface designs on the Van Der Laan five point rating scale.

The grounded slave dynamics conditions had a significantly higher usefulness score as well as significantly higher satisfaction score compared to the aerial slave dynamics ($F(1,13) = 10.7, p < 0.01$) and ($F(1,13) = 13.7, p < 0.01$) respectively. There were however no effects with regards to haptic feedback for both the usefulness score and the satisfaction score on the aerial and grounded slave conditions ($p = 0.248$) and ($p = 1$).

4 Discussion

The results indicate that different interface designs had significant effects on performance and effort of the tested remote manipulation task. More specifically, subjects were faster (67 %) with a higher average contact force (47 %) and performed less reversals (39 %) on a position control 3-DoF manipulator compared to a gamepad with rate control. However, enhancing the 3-DoF manipulator with haptic feedback from contact interaction with the environment didn't affect the performance and caused only an increase of subject effort in the z-direction (36 %) and not in the y-direction. Performing tasks in remote areas with UAVs is not ideal: Subjects performing this task with aerial slave dynamics were roughly 2.5 times slower, exerted a smaller average force (21 %) and performed roughly 2.2 times more reversals compared to performing the same task with a grounded robot. More detailed interpretations and implications of these results are discussed in the following sections.

4.1 Master device

A 3-DoF manipulator with position control allowed subjects to perform the flip of a safety switch in less time, with a higher average force, and with less steering corrections than a gamepad with rate control.

When performing small and dexterous tasks, the human hand and fingers perform exceptionally well. A position controlled manipulator functions in a manner very similar to how humans would perform the task if physically in that location. Using the two thumbsticks on a (gamepad or similar device) requires more mental effort to translate what we want the slave to do, in to the required actions we must perform; especially with the complex higher order dynamics of a UAV. Rate control resulted in more overshoots

of the target position which is why it took more time and more reversals. Additionally, a lower average contact force was realised to successfully flip the switch. This result is also supported by the measured subjective acceptance of the participants, where the gamepad with rate control was found far less useful and satisfactory than the 3-DoF manipulator with position control.

Also, in grounded telemanipulation, position control is beneficial over rate control when the manipulation space is similar to that of the human operator's control space [26], [35]. Various manipulation tasks have been performed before with UAVs but always with a gamepad or similar and in rate control [14], [12], [11], [19]. One group mentions using a haptic interface to control a UAV in contact with the environment, similar to our objective, but employs rate control [36]. Though they were able to prove their algorithm experimentally, their approach was not compared to other interfaces or controllers for any practical tasks.

It is peculiar that more advanced human-machine interfaces, proven in other fields, are not making their way into either research or the practice of aerial telemanipulation. Literature mentions the lack of good interfacing yet even a very recent survey still only mentions more traditional setups such as a joystick, keyboard, mouse, push-button panel or touchscreen. [37]. We believe that interface improvements could significantly boost the effectiveness and utilization of UAVs.

An inherent difficulty in fairly comparing these two interfaces (conditions I and II), is that two modifications were done: the master device (from bilateral to unilateral) as well as the type of controller (from rate to posiiotn control). Ideally position control does not require a steering reversal to reach a target position whereas rate control will always require one steering reversal. Though the number of reversals was still significantly lower for the manipulator, because of these two differences between the interface designs, nothing can be said about the root cause of the reduction in reversals.

In other words it is evident that the interface used for aerial telemanipulation should be at least 3-DoF manipulators that incorporates position control as this was shown to be much more effective than a gamepad with rate control.

4.2 Haptic feedback

The addition of haptic feedback to the 3-DoF manipulator did not affect any performance or effort metrics, except for the effort in z-direction, which was increased due to haptic feedback. Subjective acceptance was also not affected by having haptic feedback. Contrary to our hypothesis which stated that having haptic feedback from the contact forces would improve task performance, similar to other literature [26], [38], [39], [40]. One explanation may be that haptic feedback is simply less helpful in aerial telemanipulation then it is in grounded telemanipulation. However this is very unlikely not only based on findings of other researchers but also because grounded telemanipulation was also tested in this context and here also a significant difference was found (see section). Another, more likely explanation is that the quality of force cues is not sufficient. It has been motivated before that erroneous haptic feedback can lead to higher subject effort (like we saw in one of the two axis directions) and consequently a decreased task performance [41]. In our case, this could be due to the forces coming from Gazebo and the design of the force feedback (which was characterized by subjects as not so smooth), or due to the limited force reflecting capabilities of the Novint Falcon, chosen as a financially comparable interface to the gamepad. There are, however, much better-performing haptic manipulators available [42] which perhaps would very well lead to increasing performance of the operator.

Another possibility is that the beneficial effects of having haptic feedback were used while learning, but declined by performing 20 out of 25 repetitive trials on the same master device. It is expected that the more trials that are performed, the more accurate the subject's internal forward model becomes, making the haptic feedback simply unnecessary and possibly even a nuisance. In addition, it could be that the safety switch might not be a sufficiently representative complex task to measure the effects of haptic feedback.

In short, contradictory results have been found compared to other research, as to the benefits of haptic feedback during aerial telemanipulation. Since benefits have been proven in other studies, we believe that the fact that we did not find any significant in adding haptic feedback, is most likely due to the design and quality of the haptic feedback provided.

4.3 Slave dynamics

Performing the switch task with a grounded slave (simulating a ground robot) was substantially faster, allowed for a larger average contact force and resulted in less steering corrections, than with an aerial slave. Both usefulness and satisfaction scores were also significantly higher for the grounded slave. There was a small effect of haptic feedback on both the grounded and the aerial slave where haptic feedback slightly lengthened the completion time.

We believe the main reasons telemanipulation with an aerial slave was less effective and harder is due to its 'underactuatedness'. The dynamics of quadcopters do not allow instantaneously change of the forces acting in the horizontal plane. Once contact is made, the UAV has to tilt (roll and pitch) further to increase the contact force in that direction. The same holds for detaching itself from the object; the UAV will first have to tilt backward to move back. The relatively simple position controller did not account for this "in-contact"-effect and makes the grounded slave much for forgiving where accurate positioning with the grounded slave was not needed. This knowledge should be used to better match the operators intention and the resulting output force. Also, aerodynamic forces from the propellers, which can significantly influence the behaviour of the slave, were deactivated on the grounded slave which also influence the behaviour of the slave. The closer a UAV comes to an objects, the more the vortex wind forces disrupt the stability. Earlier we stated that the addition of haptic feedback to the manipulator showed no effect on the completion time, when it came to operating a grounded slave, haptic feedback was in fact disadvantageous. This is likely due to the same reason as mentioned before where the quality of the haptic feedback was insufficient and thus considered as disruptive.

Though the aerial slave is limited in both the force/torque that can be exerted as well as in velocity, none of these limits were not reached during the experiment. Furthermore, the aerial slave conditions are subject to degraded visual feedback (the view from the camera) because of the tilting of the UAV which makes the task somewhat easier with the grounded slave. This could easily be stabilized in future work so that the camera view would always be steady to the operator.

A possible limitation of this study is that the task might have been too elementary to perform with the grounded slave— particularly when one considers that the fastest subject performed the whole task in just 4.4 s. It is likely that the performing the same task with a ground robot with realistic dynamics and stiffness limitations would be considerably harder.

We believe that this is the first time grounded telemanipulation performance was compared to aerial telemanipulation performance. Despite the fact that research has shown UAVs to be capable of small force tasks, there has been no qualitative comparison as to how aerial telemanipulation compares to grounded telemanipulation, nor how interface design influences aerial telemanipulation performance. Though it is clear that flipping a switch is less effective with a UAV than using a grounded robot, the freedom of using a UAV likely outweighs the drawback of its limited force. There is no point in having manipulation capabilities if the task at hand can't be reached due to poor mobility of ground robots in a disrupted environment.

4.4 Implications and future work

Enriching the capabilities of UAVs to do actual physical work is interesting because of their freedom of movement— not only for manipulation tasks in disaster areas, but also for testing or maintenance of facilities in dangerous and hard-to-reach areas. The safety switch task, which requires opening a rotating cover and flilpping the switch underneath, integrates various aspects of a fine motor control task. Manipulation tasks which require larger forces are less feasible for a UAV due to its limited power, unless the UAV can ground itself onto something first. Lighter tasks such as: taking samples, adjusting controls, cleaning a surface or measuring a voltage, are tasks that are hard to automate but easy for any technician to perform if physically present at that location. Making such manipulation tasks feasible for technical staff instead of for professional UAV pilots would be a huge asset to both disaster relief teams and maintenance personnel.

Ideally, an operator would have two interfaces: each optimized towards the human for either a large or a small workspace. Certain helicopters used for fire fighting and heavy lifting (e.g. Erickson Air-crane) in fact have an extra crane operator and a seperate interface specifically for object manipulation. Though this may seem redundant, the performance increase achieved in this research argues its benefit. As for haptic feedback, although this research did not confirm its benefit, it could still prove to be useful when performing unfamiliar manipulation tasks requireing fine control of forces and moments. Further improvement of the position controller in simplifying force actuation directly in the horizontal plane would likely result in further performance increase, making it even easier for an operator to perform new and unknown tasks.

The aerial telemanipulation testing ground that was created during this research makes it possible to further develop and test various interfaces for effective telemanipulation. The modular approach using ROS allows for any interface that has a C++ API to be linked. Additionally, virtually any manipulation task can be modelled and tested in Gazebo. It is for this same reason that teams of the DARPA Robotics Challenge also use Gazebo to simulate their robots in complex environments that are publicly available online [4]. The plugins from `hector_quadrotor` also have options to add signal noise to any sensor to further analyse what additional problems could occur in he real world. As a next step, a 'real-world' physical validation experiment would be relatively straight forward, as ROS messages could easily be translated to motor commands and sent to a physical UAV.

Currently, implementation of such a system is still challenging because GPS doesn't have the nessesary coverage nor the refresh rate for highly accurate position information. Such information can be estimated from on-board vision sensors with Kalman filter-like techniques. Similarly, force information can be estimated but it remains unknown if such estimates would be sufficient to generate an accurate position or haptic feedback [43], [13].

5 Conclusion

A human factors experiment was performed with the aim of developing more understanding on how a human-machine interface should be designed for the rapid and accurate performing telemanipulation with UAVs. Various interface designs were tested to investigate how these impacted the flipping of a safety switch in a virtual environment. In particular we investigated the influance of a gamepad vs a 3-DoF manipulator , and haptic feedback vs no haptic feedback, where aerial and grounded robots were compared.

In conclusion, this study shows that aerial telemanipulation can be employed for small force tasks in human-built environments. More specifically, for the experimental conditions studied, it can be concluded that for performing remote manipulation tasks with an Unmanned Aerial Vehicle:

- $\Rightarrow$ a 3 degree-of-freedom manipulator with position control substantially outperforms a gamepad with rate control
- $\Rightarrow$ haptic feedback did not influence the task performance
- $\Rightarrow$ a grounded slave substantially outperforms an aerial slave. most likely due to the underactuatedness and complex dynamics of the aerial slave.

This study shows the potential of using Unmanned Aerial Vehicles to perform human-in-the-loop remote manipulation tasks in human-built environments. Furthermore it shows that interface design has a significant impact on the performance of such aerial telemanipulation.

References

[1] J. Casper and R. Murphy, "Human-robot interactions during the robot-assisted urban search and rescue response at the World

Trade Center." *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, vol. 33, no. 3, pp. 367–85, jan 2003.

[2] R. R. Murphy, "A decade of rescue robots," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5448–5449, oct 2012.

[3] "Center for Robot-Assisted Search and Rescue (CRASAR)." [Online]. Available: http://www.crasar.org/

[4] "DARPA Robotics Challange." [Online]. Available: http://www.theroboticschallenge.org/

[5] "RoboCup 2015." [Online]. Available: http://www.robocup2015.org/

[6] "European Robotics Challange." [Online]. Available: http://www.euroc-project.eu/

[7] "Long-Term Human-Robot Teaming for Robot Assisted Disaster Response." [Online]. Available: http://www.tradr-project.eu/

[8] A. Ollero and L. Merino, "Control and perception techniques for aerial robotics," *Annual Reviews in Control*, vol. 28, no. 2, pp. 167–178, 2004.

[9] F. Matsuno and S. Tadokoro, "Rescue Robots and Systems in Japan," *2004 IEEE International Conference on Robotics and Biomimetics*, pp. 12–20, 2004.

[10] P. E. I. Pounds and A. M. Dollar, "Aerial Grasping from a Helicopter UAV Platform," *Experimental Robotics*, vol. 79, pp. 269–283, 2014.

[11] M. Orsag, C. M. Korpela, and P. Y. Oh, "Modeling and Control of MM-UAV: Mobile Manipulating Unmanned Aerial Vehicle," *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1-4, pp. 227–240, aug 2012.

[12] P. E. I. Pounds and A. M. Dollar, "UAV rotorcraft in compliant contact: Stability analysis and simulation," *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2660–2667, sep 2011.

[13] D. Mellinger, Q. Lindsey, M. Shomin, and V. Kumar, "Design, modeling, estimation and control for aerial grasping and manipulation," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2668–2673, 2011.

[14] A. Albers, S. Trautmann, T. Howard, M. Frietsch, and C. Sauter, "Semi-autonomous flying robot for physical interaction with environment," *2010 IEEE Conference on Robotics, Automation and Mechatronics*, pp. 441–446, jun 2010.

[15] S. Bellens, J. De Schutter, and H. Bruyninckx, "A hybrid pose / wrench control framework for quadrotor helicopters," *2012 IEEE International Conference on Robotics and Automation*, pp. 2269–2274, may 2012.

[16] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, "A platform for aerial robotics research and demonstration: The Flying Machine Arena," *Mechatronics*, vol. 24, no. 1, pp. 41–54, feb 2014.

[17] M. Orsag, C. M. Korpela, S. Bogdan, and P. Oh, "Valve turning using a dual-arm aerial manipulator," *2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings*, pp. 836–841, 2014.

[18] J. L. Scholten, M. Fumagalli, S. Stramigioli, and R. Carloni, "Interaction control of an UAV endowed with a manipulator," *2013 IEEE International Conference on Robotics and Automation*, pp. 4910–4915, may 2013.

[19] G. Jiang and R. Voyles, "Hexrotor UAV platform enabling dextrous interaction with structures-flight test," *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6, oct 2013.

[20] "Airobots: Innovative Aerial Service Robots for Remote Inspection by Contact," http://airobots.ing.unibo.it/, 2013. [Online]. Available: http://airobots.ing.unibo.it/

[21] "Aerial Robotics Competition." [Online]. Available: http://www.aerialroboticscompetition.org

[22] "Aerial Robotics Cooperative Assembly System." [Online]. Available: http://www.arcas-project.eu/

[23] "Aeroworks: Collaborative Aerial Workers." [Online]. Available: http://www.aeroworks2020.eu/

[24] J. Chen, E. Haas, and M. Barnes, "Human performance issues and user interface design for teleoperated robots," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 6, pp. 1231–1245, 2007.

[25] J. Scholtz, J. Young, J. L. Drury, and H. A. Yanco, "Evaluation of human-robot interaction awareness in search and rescue," *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, pp. 2327–2332 Vol.3, 2004.

[26] S. E. Salcudean, M. Zhu, W.-H. Zhu, and K. Hashtrudi-zaad, "Transperent Bilateral Teleoperation under Position and Rate Control," *The International Journal of Robotics Research*, vol. 19, no. 12, pp. 1185–1202, 2000.

[27] R. R. Murphy and J. L. Burke, "Up from the Rubble: Lessons Learned about HRI from Search and Rescue," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 49, no. 3, pp. 437–441, 2005.

[28] T. Lam, "Haptic Interface for UAV Teleoperation," PhD dissertation, Delft University of Technology, 2009.

[29] B. Weber and C. Eichberger, "The Benefits of Haptic Feedback in Telesurgery and Other Teleoperation Systems: A Meta-Analysis," in *Universal Access in Human-Computer Interaction. Access to Learning, Health and Well-Being*. Springer, 2015, pp. 394–405.

[30] M. K. OMalley, A. Gupta, M. Gen, and Y. Li, "Shared Control in Haptic Systems for Performance Enhancement and Training," *Journal of Dynamic Systems, Measurement, and Control*, vol. 128, no. 1, p. 75, 2006.

[31] "hector_quadrotor - ROS wiki." [Online]. Available: http://wiki.ros.org/hector{_}quadrotor

[32] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. Von Stryk, "Comprehensive simulation of quadrotor UAVs using ROS and Gazebo," *Simulation, Modeling and Programming Autonomous Robots (Lecture Notes in Computer Science)*, vol. 7628 LNAI, pp. 400–411, 2012.

[33] V. Kumar and N. Michael, "Opportunities and challenges with autonomous micro aerial vehicles," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1279–1291, 2012.

[34] J. D. Van Der Laan, A. Heino, and D. De Waard, "A simple procedure for the assessment of acceptance of advanced transport telematics," *Transportation Research Part C: Emerging Technologies*, vol. 5, no. 1, pp. 1–10, 1997.

[35] W. K. W. Kim, F. Tendick, S. Ellis, and L. Stark, "A comparison of position and rate control for telemanipulations with consideration of manipulator system dynamics," *IEEE Journal on Robotics and Automation*, vol. 3, no. 5, pp. 426–436, 1987.

[36] A. Y. Mersha, S. Stramigioli, and R. Carloni, "Bilateral teleoperation of underactuated unmanned aerial vehicles: The virtual slave concept," *2012 IEEE International Conference on Robotics and Automation*, pp. 4614–4620, may 2012.

[37] J. M. Peschel and R. R. Murphy, "Handbook of Unmanned Aerial Vehicles," in *Handbook of Unmanned Aerial Vehicles*. Springer Netherlands, 2015, pp. 2389–2403.

[38] B. Hannaford, L. Wood, B. Guggisberg, D. McAffee, and H. Zak, "Performance Evaluation of a Six , Axis Generalized Force-Reflecting Teleoperator," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, no. 3, pp. 620–633, 1991.

[39] R. L. Klatzky, S. J. Lederman, and C. Reed, "There's more to touch than meets the eye: The salience of object attributes for haptics with and without vision." *Journal of Experimental Psychology: General*, vol. 116, no. 4, pp. 356–369, 1987.

[40] J. G. Wildenbeest, R. J. Kuiper, F. C. van der Helm, and D. A. Abbink, "Position Control for Slow Dynamic Systems: Haptic Feedback Makes System Constraints Tangible," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2014, pp. 3990–3995.

[41] G. A. V. Christiansson, "Hard Master, Soft Slave Haptic Teleoperation," PhD disseration, Delft University of Technology, 2007.

[42] P. Lambert and J. Herder, "A Novel Parallel Haptic Device with 7 Degrees of Freedom *," in *World Haptics Conference (WHC), 2015 IEEE*, vol. 31, 2015, pp. 183–188.

[43] F. Augugliaro and R. D'Andrea, "Admittance control for physical human-quadrocopter interaction," *European Control Conference*, pp. 1805–1810, 2013.

Appendix A

Quick start guide

This chapter has three section. The first describes how to install all the necessary software to run the simulation. The second explains how to actually run the simulation if it is installed correctly. The final section mentions some known issues and bugs that I encountered during using this software. Other software for data acquisition and analysis is also needed but will not be explained in this section. You can read about the implementation of this software in chapter B. This guide will use the following pieces of software:

1. Ubuntu 14.04 LTS (Trusty)
2. ROS Indigo Igloo
3. Gazebo 5.1
4. QtCreator 3.0.1
5. Matlab R2015a

If you have followed this guide and installed all the necessary files and folders, you should have an exact copy of my project. Throughout this thesis, certain files and folders will be described or used in a section. The location of these files will be shown at the top of a section, as follows:

Project location

```
~/catkin_ws/src/[A project folder]
```

A.1 Installation

A.1.1 Ubuntu, ROS and Gazebo

This installation manual does assume some minor software knowledge. If you have no experience with Linux and/or C++ I suggest you focus on that first. There are tons of tutorials on learning the basics online to bash (the Unix/Linux terminal) and C++ or python. Next, if you're totally new to ROS, a good starting point is the: "A Gentle Introduction to ROS" (AGITR) which can be found online [1]. Another good guide is "ROS 101" by Clear Path Robotics [2]. If you follow these guides successfully, you should end up with Ubuntu, ROS and Gazebo 2.1 installed. As for any answers to questions you have along the way, it's unlikely that you're the first to have asked them. ROS has a very good forum which is also used by developers all over the world where you can also post your new questions [3]. The same holds for Gazebo [4].

A.1.2 Installing hector_quadrotor

There are a few ways to install the `hector_quadrotor` package but the important thing is, you need to install it from source. I would suggest using git as opposed to rosinstall, but they will both work. First `cd` into your catkin workspace, and then clone the git repository:

```
$ cd ~/catkin_ws
$ cd src
$ git clone https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor.git
```

Next compile your catkin workspace. This is a returning command and will from here on be referred to simply as "compile your workspace":

```
$ cd ~/catkin_ws
$ catkin_make
```

You can now spawn a quadrotor in an empty world except you will have no way to control it. To test if it all runs though, you can start it like this:

```
$ roslaunch hector_quadrotor_gazebo quadrotor_empty_world.launch
```

A.1.3 Adding my work

The nice thing about ROS is that it is all so modular that I can easily create my own catkin packages that work alongside those of `hector_quadrotor`. In theory this would mean we could both keep working on our packages independently and everything would still work. In reality though, as you can imagine, this is a lot more complex.

Next we will copy all the catkin packages I created to your catkin workspace, add some additional files (controllers) to the `hector_quadrotor` package, add model files to the Gazebo home folder and copy the data acquisition scripts. I created bash scripts for both these steps which you can extracted and run as follows:

```
$ cd ~/catkin_ws/src
$ unzip UAV_tele_pkgs
$ ./add_new_pkgs
$ ./merge_hector_pkgs
$ ./add_gazebo_models
$ ./data_acq
```

A.1.4 Manipulator setup

The Novint Falcon has an open source driver called `libnifalcon` which needs to be installed for it to be recognized by Ubuntu [5]. This driver provides an API to access features of the Falcon such as reading encoder positions, reading end effector positions and prescribing forces. First extract these files to your workspace

```
$ cd ~/catkin_ws/src
$ tar xvf libnifalcon-1.0.1.tar.gz
```

Next compile the files per instructions inside the package. I do this inside my workspace but you can compile this anywhere as it uses a different compiler then our catkin workspace. Do the following:

```
$ cd ~/catkin_ws/src/libnifalcon
$ mkdir build
$ cd build
$ cmake -G "Unix Makefiles" ..
$ make
$ make install
```

We have no compiled the falcon driver into a `build` folder under the `libnifalcon` folder. The library files (`*.so`) however need to be in a place where ubuntu can find them, which is not in your workspace. Therefore copy them to your local library folder:

```
$ cd ~/catkin_ws/src/libnifalcon/build/lib
$ sudo cp *.so /usr/local/lib/
```

Check to make sure the falcon is working by running the `findfalcons` binary in the build folder. This usually needs a few tries (I honestly don't know why...) and when it succeeds it should show you the raw encoder positions of the 3 arms:

```
$ cd ~/catkin_ws/src/libnifalcon/build/bin
$ ./findfalcons
```

A.1.5 Gamepad setup

Fortunately Ubuntu and ROS already include all the necessary items to use pretty much any standard USB gamepad. For compatibility check `http://wiki.ros.org/joy`. One handy tool though is `jstest-gtk` which gives you a GUI for joystick setup and calibration. Install it by typing the following, after which it should appear on your Ubuntu task bar:

```
$ sudo apt-get install jstest-gtk
```

You can use this tool to calibrate the gamepad's thumbsticks and to see which device name it has been assigned. The default is `/dev/input/js0` but some mouses or other input devices appear in this list as well which could mean it is assigned a different name. The package `hector_quadrotor_teleop` provides a ROS node to control the UAV with a gamepad. This is also where you set the correct device. You can set and change the current device by for instance opening the file `xbox_controller.launch` in the package folder and editing the `joy_dev` line:

```
$ cd ~/catkin_ws/src/hector_quadrotor/hector_quadrotor_teleop/launch/
$ nano xbox_controller.launch
```

A.1.6 Update Gazebo

Gazebo 2.1 was already quite outdated when I started and does not provide a lot of features that the current stable did. Alternative physics engines, extensive stiffness, damping, and friction settings were not available yet. Therefore I decided to migrate the whole `hector_quadrotor` package to gazebo5. Again this has to be done from source as opposed to just replacing the binary files from gazebo2 with those from gazebo5.

To do this, remove gazebo2 the packages that connect it to ROS and install gazebo5. The following commands in a terminal window:

```
$ sudo apt-get remove ros-indigo-gazebo-ros-pkgs
$ sudo apt-get remove gazebo2
$ sudo apt-get install gazebo5
$ sudo apt-get install ros-indigo-gazebo5-ros-pkgs ros-indigo-gazebo5-ros-control
```

Once you have done this, you will have to compile your workspace with the new version of Gazebo. This will probably throw you some errors about missing or faulty packages. Look carefully at what the errors are and try and fix them with some online help (The ROS and Gazebo forums are very good for this). In my case it was `ros_control` which was now needed from source. I installed it like this:

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ros-controls/ros_control.git
```

Again compile your workspace until everything succeeds.

A.2 Run

This section presumes you have not only a working copy of ROS Indigo and Gazebo 5.1 but also all my other packages (`quad_world`, etc.) copied and compiled, the changed files (from `hector_quadrotor`) copied and compiled,the Novint Falcon drivers working and the gamepad working. Great! Now there are two ways to start the whole simulation: through the command line (the usual ROS way, from the command line) or through my Experiment Graphical User Interface (GUI) A.1). I created the GUI because starting a simulation with 13 different arguments every time, from the command line, is painful. You can set defaults but it still doesn't keep you from forgetting to change an argument. The GUI is written in C++ in a Qtcreator IDE. The source files are located in the folder shown above.

A.2.1 Command line

Project location

`~/catkin_ws/src/quad_world/launch`

A ROS launch file called `start.launch` sets all the parameters, loads all the necessary components and then starts the simulation. Open it and you will see the following settings at the top of the launch file:

```
<arg name="keyb" default="true" />
<arg name="joy" default="false" />
<arg name="falcon" default="false" />
<arg name="HF" default="false" />
<arg name="fpv" default="false" />
<arg name="gui" default="true" />
<arg name="record" default="false" />
<arg name="plot" default="false" />
<arg name="world_name" default="$(find quad_world)/worlds/safety_switch3.world" />
<arg name="y_offset" default="0" />
<arg name="ctrltype" default="controller/twist" />
<arg name="nodyn" default="false" />
<arg name="bagfilename" default="pilot" />
```

Argument name	Valid values	Comments
`keyb`	`true` or `false`	If set to `true`, the keyboard keys will be the active interface. Useful for testing. Cannot be used with `falcon` or `HF`.
`joy`	`true` or `false`	If set to `true`, the gamepad will be the active interface. Cannot be used with `falcon` or `HF`.
`falcon`	`true` or `false`	If set to `true`, the Novint Falcon will be the active interface. Cannot be used with `keyb`, `joy` or `HF`.
`HF`	`true` or `false`	If set to `true`, the Novint Falcon with haptic feedback will be the active interface. Cannot be used with `keyb`, `joy` or `falcon`.
`fpv`	`true` or `false`	If set to `true`, the application `Rviz` will start the view from the Gazebo camera on top of the UAV.
`gui`	`true` or `false`	If set to `true`, the Gazebo interface will load. If not, Gazebo will run headless which is useful for only viewing the camera (like during experiments).
`record`	`true` or `false`	If set to `true`, the topics marked in the launch file `start.launch` will be recorded to a `.bag` file and saved in the folder `/.ros/`.
`plot`	`true` or `false`	If set to `true`, the application `rqt_plot` will start to plot signals/messages straight from ROS topics.
`world_name`	see Tab. A.2	Sets which world Gazebo will load and where the UAV will spawn into. Can be any `.world` file located in a catkin package. All the world listed in Tab. A.2 have to be preceded by `"$(find quad_world)/worlds/"`. For example: `"$(find quad_world)/worlds/safety_switch3.world"`
`y_offset`	any `double`	The distance at which the UAV will spawn from the centerline to the left/right. If you are using the landing pad then this value must be $-0.2 < y_{\text{offset}} < 0.2$ (in meters).
`ctrltype`	see Tab. A.3	Sets the type of controller that should be started when spawnen the UAV. See Tab. A.3 for options and definition.
`nodyn`	`true` or `false`	If set to `true`, the aerodynamic (drag and propeller) forces are not loaded. This is especially useful if you want to simulate a grounded robot and are using the fully actuated controller.
`bagfilename`	any "`string`"	If `record` is `true` then a filename can be specified here under which the `.bag` file should be saved. The resulting file will have the following name: `[bagfilename]2015-12-25-09-33-48.bag`

Table A.1: List of arguments and their valid values that are passed through the launch file that starts the simulation.

World name	Comments
`empty_world.world`	An empty world with only a light and a ground plane.
`wall.world`	A world with only a wall with default gazebo stiffness $k = 1e8$ [N/m].
`wall_table.world`	A world with a wall, a table and a coke can on the table. The table has stiffness $k = 10000$ [N/m] and damping $b = 1$ [Ns/m]. The wall and coke can have default stiffness.
`wall_button.world`	A world with a wall and a springy button. Button has prismatic joint stiffness $k = 1000$ [N/m] and damping $b = 1$ [Ns/m].
`button.world`	A world with a floating button. Button has a prismatic joint stiffness $k = 1000$ [N/m] and damping $b = 1$ [Ns/m].
`lever.world`	A world with a floating lever with a revolute joint. The joint has damping $b = 0.1$ [Nms/rad] and static friction $F_f = 0.3$ [Nm].
`drc.world`	A world with multiple objects in it from the DARPA Robotics Challenge like a wheel valve, a ball valve and a heavy door.
`material_changer.world`	A world with a test object for the `model_material` gazebo plugin.
`dirty_wall2.world`	A world with the second version of the dirty wall. This wall has just one wall segment that changes color depending on the interaction force applied. See `dirty_wall` plugin in section B.4.3 for more details.
`pillar.world`	A world with a pillar with $k = 100000$ [N/m].
`safety_switch3.world`	A world with a floating safety switch. See `safety_switch` plugin in section B.1.1 for more details.
`dirty_wall3.world`	A world with a grid of 3x3 dirty wall elements. See `dirty_wall` plugin in section B.4.3 for more details.
`training.world`	A world that has 3 transparent spherical targets floating just above and to the side of the spawning location of the UAV. Used during the experiment to get subject accustomed to the interface.

Table A.2: List of worlds that I created that can be loaded into Gazebo where the UAV can spawn into.

Controller name	Comments
`controller/twist`	Rate controller
`controller/pose controller/twist`	Position controller
`controller/pose_nodyn controller/twist_nodyn`	Grounded slave position controller
`controller/twist_nodyn`	Grounded slave rate controller
`controller/twist_joy`	Gamepad rate controller

Table A.3: List of possible controllers that the UAV can have. More information about these controllers can be found in section B.2

Once you have set the parameters you want, start the simulation as follows:

```
$ roslaunch quad_world start.launch
```

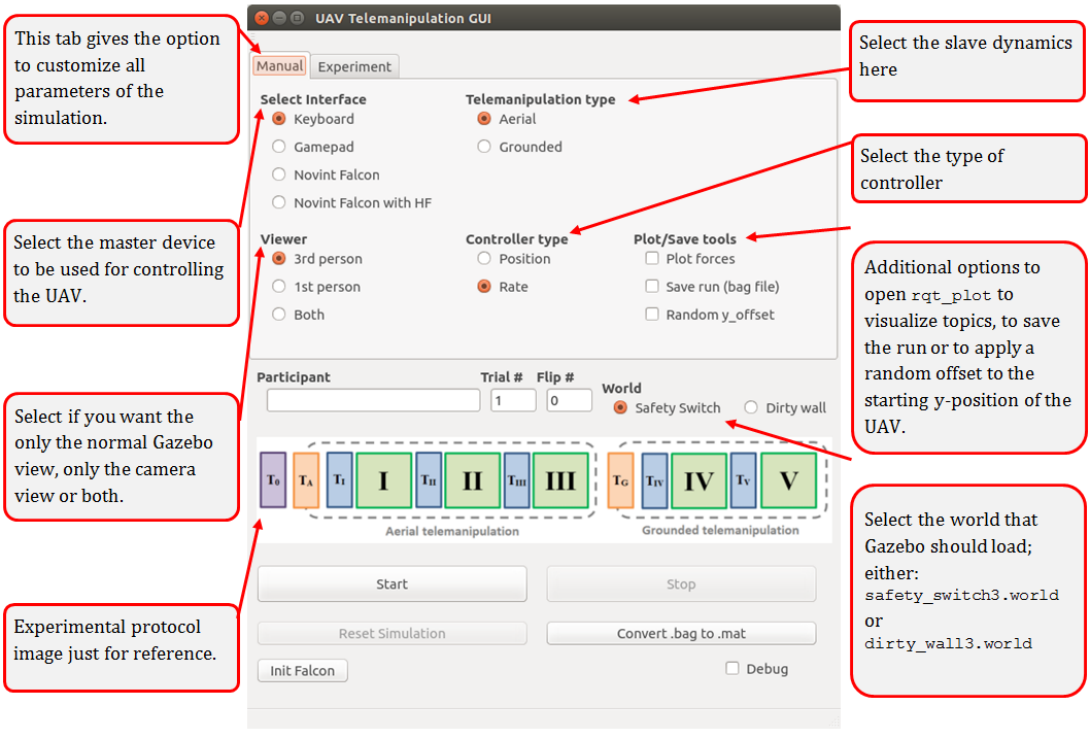
A.2.2 GUI

Project location

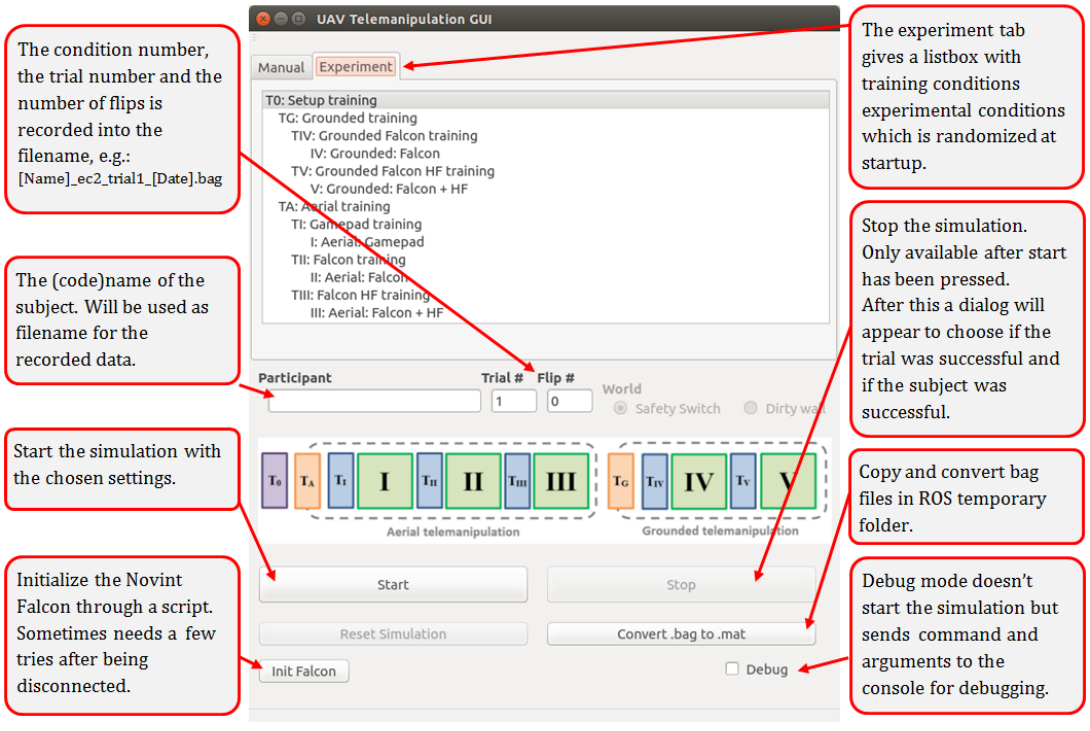
`~/catkin_ws/src/gui/UAV_tele_GUI`

The GUI works by starting a new `xterm` session using the launch file `start_clean.launch` which is located in the same folder as `start.launch`. The GUI sets the required parameters and as soon at the `Start` button is pressed, the simulation is started with all the parameters set at that time. Unfortunately I was not able to make the GUI run without running it from QtCreator because the environmental variables for ROS are then missing. Therefore you have to compile and run (▶) the GUI project (`UAV_tele_GUI.pro`) directly from QtCreator for it to in turn launch the simulation.

The GUI has two tabs: one for manual control and testing (Figure A.1a) and the other for experimental control (Figure A.1b). The functions of all the objects in the GUI and how to actually start the simulation is explained below (Figure A.1).



(a) Manual tab



(b) Experiment tab

Figure A.1: Graphical User Interface (GUI) to start the simulation

A.2.3 Known issues and bugs

The simulation software, GUI to start the simulation and the plugins have a number of issues and known bugs. Most of these bugs occur sporadically and thus seem to be timing based. They occur but not often enough for me to make it worth while to make big efforts to fix them. It is likely some of theme are bugs from Gazebo itself and will be fixed in future versions. The following list of issues and bugs is known:

1. The simulation sometimes crashes even before it starts correctly. One of the Gazebo components will then not open; mostly the user interface of Gazebo itself. Because I was unable to find the cause, I ended up adding a prompt window when the `Stop` button is pressed in the GUI which gives me the option to mark a trial as failed. In this case the trial number and/or the flip number will not be increased by 1.

2. When loading the dirty wall model, the simulation often crashes. The dirty wall model (in the end not used in this experiment) consists of a grid (small surface) of 5×5 of the same model. I believe this might be the limit of the amount of links that can be spawned under one model. Also, because the `dirty_wall` plugin runs on each of these links, it makes the model even more computationally demanding. Limiting the number to for instance 3×3 significantly reduced the number of crashes during startup.

3. It is mentioned before, but I was unable to run the GUI I created (Fig A.1) directly from Ubuntu. It has to be started from QtCreator itself. This is because the environmental variables for ROS are not loaded without QtCreator. There are ways to do this but since the GUI runs fine from QtCreator, I didn't spend too much time trying to fix this.

4. When the Novint Falcon is connected, all three colors will light up on the interface. Though it may seem like it is connected, it is not. I was unable to find what part of code needs to be run to initialize it but I did find a workaround. The script `findfalcons` located in the folder: `~/catkin_ws/src/libnifalcon/build/bin` does connect the falcon correctly. It always fails the first time but something happens after it fails that makes it work the second time. When it is successful, the light will flash all three colors and the raw encoder positions will run through the terminal for 3 seconds. After it is done, the light on the falcon with become solid red. This means it is successfully connected.

Appendix B

Software implementation

Disclaimer: My major is Mechanical Engineering, not Software Engineering. I more or less taught myself: Linux, C++, Python, ROS and Gazebo for my master thesis project. All the software I wrote and/or adapted is built to primarily work. Some of it therefore might not be consistent, tidy, follow common practice or efficient.

Using modular software package like ROS is great but it has one downside: It quickly becomes very very complex. All the ROS launch files can spawn new launch files which independently then in turn spawn new ROS nodes, each with their own configuration. And then there's Gazebo that runs independently and also runs various plugins that can change the behaviour of the models in the virtual world. These Gazebo plugins can then also interact with ROS making everything turn into one big spaghetti. The graph below attempts to clarify how the simulation works and which process runs which (Figure B.1).

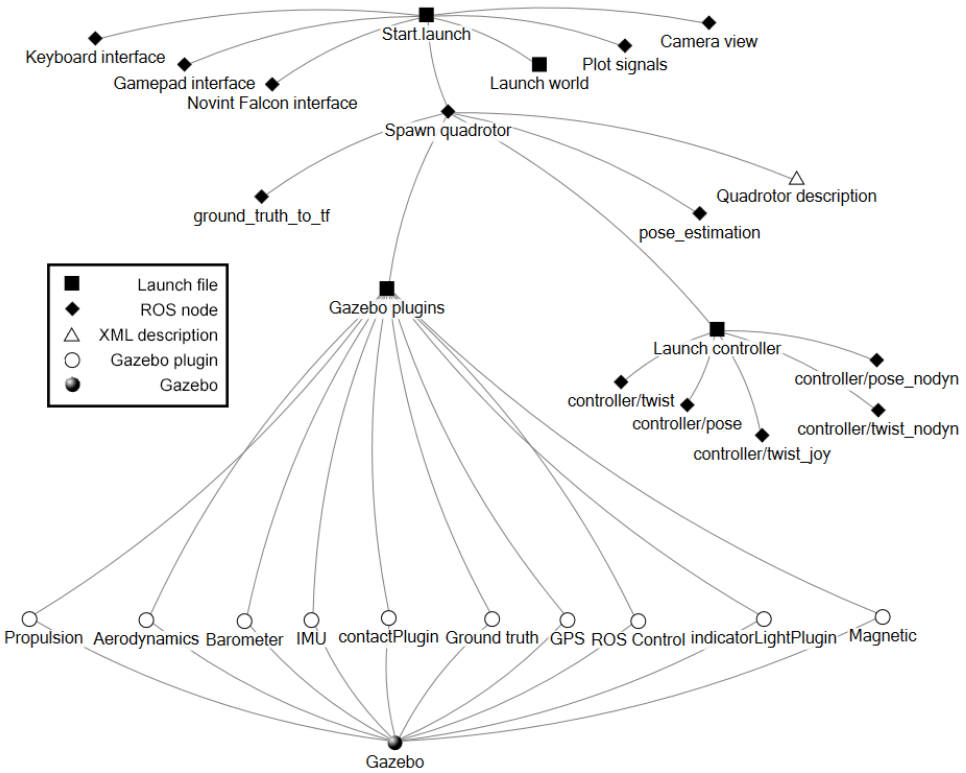


Figure B.1: A schematic overview of how the simulation implementation works

This chapter discusses various parts of the software used for the human factors experiment. The following components are discussed: Manipulation tasks that were used are discussed, controllers used to steer the UAV, haptic feedback design, Gazebo plugins, ROS nodes and finally data acquisition software.

B.1 Task design

The choice of an appropriate manipulation task to test various interface designs has a challenging trade off. The research will be more useful in practice if a larger variation of tasks is tested however the requirements for each task is so different that the design is likely to become to specialized for a certain task. At the start of this research, the idea was to create 3 different manipulation tasks in simulation that span a large task space, in both the *fine motor control tasks* as well as the *tool assisted tasks* that require a relatively small force/torque level. Tasks like simple contact with an object to measure the voltage were deemed too simple as they only require contact and not actual manipulation whereas large force tasks such as opening doors were considered unrealistic for the time being. The chosen tasks lie somewhere in the middle between these two, marked in red (Figure B.2).

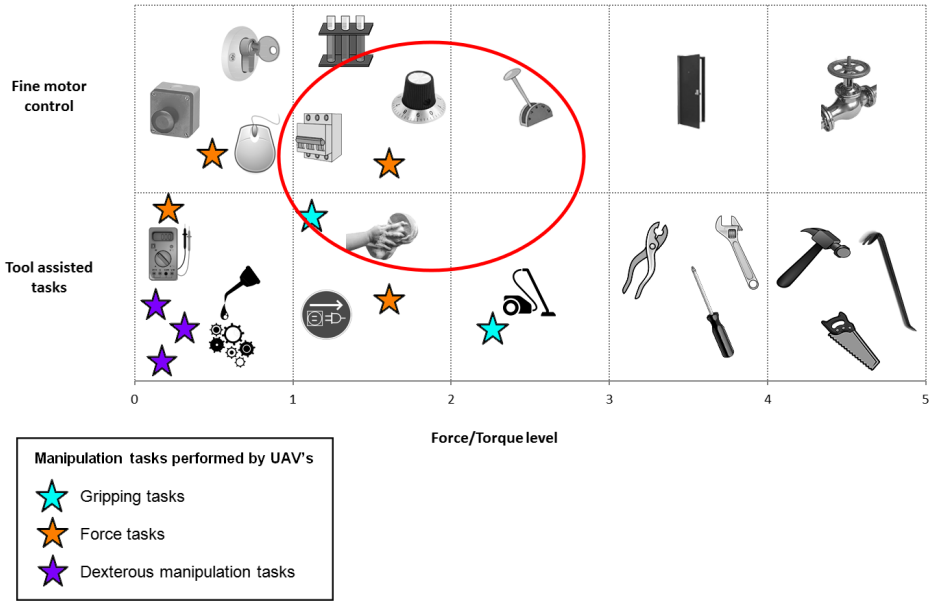


Figure B.2: Task taxonomy including both potential manipulation tasks in disaster areas (indicated by symbolic figures) as well as categorized tasks that have been performed by UAVs in literature (indicated by stars). The red area marks the area of focus from which 3 manipulation tasks were distilled.

The resulting tasks, which are relevant disaster area tasks, were chosen to be the following: A safety switch that had to be flipped, a lever that had to be activated and a small surface that had to be cleaned (Figure B.3). Shortly after the idea to incorporate a lever like "rotating manipulation" into the safety switch to limit the amount of tasks to just two. This was done by adding a cover onto the safety switch which had to be opened first before the switch can be flipped.

(a) A safety switch (b) An industrial lever (c) A dirty gauge

Figure B.3: The intial manipulation tasks that were to be used to test interface design effectiveness. The second (lever) task was later eliminated to because the cover of the safety switch already resembled a comparable task to that of the lever.

B.1.1 Safety switch

Project location

```
~/catkin_ws/src/quad_world/worlds/safety_switch3.world
~/.gazebo/models/safety_switch
```

The safety switch task is a very relevant task in a technological disaster scenario. Every industrial area has control rooms and interfaces everywhere designed for humans to control various systems such as valves and emergency stops. Safety switches exist in a large variety of shapes and sizes. One of the more common ones is the one shown above (Figure B.3b).

Initially I tried to create the cover myself but later realised an online 3D model would probably be more correct. In the end I downloaded one from GrabCAD called toggle-switch-guard [6]. FreeCAD and then Blender were used (both free to download) to convert the CAD file into a .dae file which can be loaded into a Gazebo model file as a mesh.

The switch itself was modelled as a cylinder with diameter of 10 mm and a length of 5 cm. Two revolute joints were added to the model: one at the back of the switch cover and one at the base of the switch. Though the actual sizes of such switches vary, this is a little less than twice the size of most industrial switches. The cover and switch were set to have a mass of respectively 20 g and 10 g. The joint dynamics of the cover, which created the most difficulty of the task, were set to have a damping of $0.05\,\mathrm{N\,m\,s\,rad^{-1}}$ and a static friction of $0.02\,\mathrm{N\,m}$. The switch had the same damping but half the static friction, $0.01\,\mathrm{N\,m}$. These values were chosen experimentally to make the task difficult, but if the UAV was positioned correctly, not impossible.

The result was the model shown below (Figure B.4).

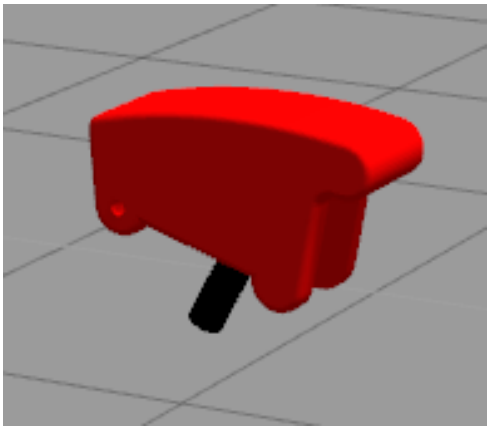


Figure B.4: Safety switch model. The cover of the switch first had to be lifted up, before the switch underneath could be switched.

B.1.2 Dirty wall

Project location

```
~/catkin_ws/src/quad_world/worlds/dirty_wall3.world
~/.gazebo/models/dirty_element2
```

After or during a technological disaster, it is likely that structures have collapsed and that there is rubble and dust everywhere. It could be that critical systems are unstable and therefore need to be monitored by viewing displays, screens or gauges but are covered in dust or debris. The second task therefore involves cleaning a small surface, like for instance a pressure gauge, to then make reading the values of the gauge possible.

Because simulating such an effect would be very difficult (both physically and visually), a simple approach was taken to take a small square surface, split it into smaller square sections and then have each section change color depending on the contact with the UAV. To approximate the removing of debris, the normal force on the surface was used, multiplied by the time it was in contact and by a constant

K. This gave a relatively simplistic but effective effect: The longer the UAV was bumped up against the surface, the more material was removed or in this case, the more the model element changed color (From black to red). This resulted in the following model shown below (Figure B.5).

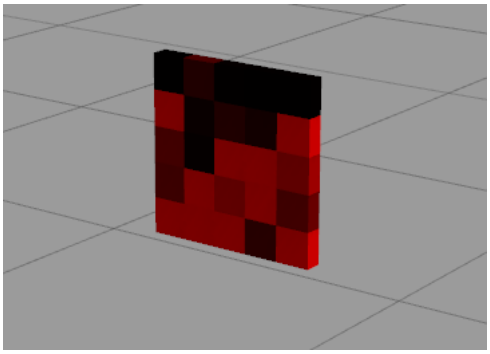


Figure B.5: Dirty wall model. The more the model swiped around the surface of the dirty wall, the more elements would change color.

Later a more realistic relation was implemented where the removed particles is dependent on primarily the sliding force and the distance: the Archard Equation:

$$Q = \frac{K \cdot W \cdot L}{H} \tag{B.1}$$

Where Q is the total volume of the wear produced [m^3], K is a dimensionless constant, W is the total normal load [N], L is the sliding distance [m] and H is the hardness of the softest material [HB] (more about this implementation in section B.4.3).

Later, just like with the lever task, the dirty wall task was also eliminated from this research because, with five different interface designs, the experiment would be too long. This was decided shortly after the pilot experiment, which was only done with the safety switch, had 3 repetition trials and already took about an hour without questionnaire. In addition, the focus of this experiment was to research the interface designs as independent variable and not the environment.

Though, as mentioned above, this task was not used for the actual experiment, it is still a very interesting task to practice. Since there are so many different types of surfaces at different angles with different sides, it is important to have a effective way of cleaning them.

B.1.3 DRC tasks

Project location

`http://models.gazebosim.org/`

The nice thing about Gazebo is that so many people are working on improving the simulation quality and the models that can be used. Even better is that a lot of teams competing in the DARPA Robotics Challenge (DRC) also use ROS for their framework and Gazebo for their simulations. Because of this, a database of DRC practice models including standardized models for doors, valves and rubble are available online. There are two ways to access these models: Through the interface of Gazebo and directly online. In the Gazebo interface, click the `Insert` tab and wait for the list at the bottom to populate (you need an internet connection). After this you should be able to directly download and insert DRC models. The second option is to download the models manually by browsing to the location stated above.

B.2 Controller design

Project location

```
~/catkin_ws/src/hector_quadrotor/hector_quadrotor_controller
~/catkin_ws/src/hector_quadrotor/hector_quadrotor_controller/original
~/catkin_ws/src/hector_quadrotor/hector_quadrotor_controller/param
~/catkin_ws/src/hector_quadrotor/hector_quadrotor_controller/param/original
```

The controllers that we're developed for this experiment are modified controllers from the original `hector_quadrotor` package (location shown above). These cascaded PID-controllers are explained in detail in a scientific paper describing the original ROS package [7]. The changes are motivated and discussed per controller in the following sections.

B.2.1 Rate controller

Project location

```
~/catkin_ws/src/hector_quadrotor/hector_quadrotor_controller/param/controller.yaml
~/catkin_ws/src/hector_quadrotor/hector_quadrotor_controller/src/twist_controller_joy.cpp
```

The rate controller was only used for the gamepad interface. Two changes were made to this controller:

1. The integral action of the PID controller was set to 0
2. The yaw angle (z-rotation) is governed by an additional controller

The first change is simple and just requires a change in the controller parameters in the `param` folder. Both the linear and the angular integral action K_i were changed to 0. During normal free-flying, you want the UAV to reach the set velocity accurately in steady state which is why the integral of the error is in this case useful. During interaction however, we want the UAV to stay at a banked level when in contact with the environment if all set velocities are 0. Because the UAV is underactuated in the horizontal plane, the rate controller always controls the velocity of the UAV to 0 with he assumption it should be level. By changing the integral actions we could prevent the UAV from tilting further when in contact.

For the second change, we want the yaw angle always to control itself to 0; independent of external forces. When the UAV comes in contact with objects, the yaw angle can change due to external forces and since only the velocity is governed, the additional yaw angle is not corrected. To do this, an extra negative feedback P-controller was added onto the PID controller that controlled the torque in z-direction:

```
wrench_.wrench.torque.z = inertia_[2] * pid_.angular.z.update( command.angular.z,
↪  twist.angular.z, 0.0, period) + Kp * ( - pose_->get()->orientation.z );
```

The chosen value for the gain was $K_p = 1$, which was tested experimentally. Since the forces are relatively small, the correction doesn't have to be enforced with a very high gain. The forces are also not large enough that a derivative action is needed. The steady state error is also practically non existent because once the UAV is free from contact again, nothing is keeping it from moving back to its zero yaw angle.

B.2.2 Position controller

Project location

```
~/catkin_ws/src/hector_quadrotor/hector_quadrotor_controller/param/controller.yaml
```

For this controller only the controller gains were slightly adjusted. These changes were done experimentally and resulted in very little change in behaviour.

B.2.3 Grounded slave controller

Project location

```
~/catkin_ws/src/hector_quadrotor/hector_quadrotor_controller/pose_controller_nodyn.cpp
~/catkin_ws/src/hector_quadrotor/hector_quadrotor_controller/twist_controller_nodyn.cpp
```

To compare performance with a grounded robot, I needed a grounded slave. I thought about creating or using a different robot in Gazebo as slave but not only would this be a lot of work, it would also induce other differences between the conditions. Therefore I decided to create a "grounded slave" controller for the UAV which was fully actuated. Normally the movement in the horizontal plane was a result from the UAV tilting but by changes the controller, but after the changes, the UAV could exert forces directly in the horizontal plane. Because the PID work in cascaded form (the position is translated to a velocity which is translated to an acceleration) both the rate and the position controller have to be altered.

The rate controller (`twist_controller`) was adjusted by changing the controlled forces in x and y direction from constant 0 to Newton's second law ($m \cdot a$), as follows:

```
wrench_.wrench.force.x  = mass_ * (acceleration_command.x * load_factor);
wrench_.wrench.force.y  = mass_ * (acceleration_command.y * load_factor);
```

The position controller `pose_controller` was adjusted by adding a section that controlled the pitch and roll angle. Without this, the UAV would roll or pitch into a certain direction after being in contact with the environment. The change was done as followS:

```
// control pitch angle
output.angular.x =
 ↪  pid_.pitch.update(HeadingCommandHandle(*pose_input_).getError(*pose_),
 ↪  twist_->twist().angular.x, period);

// control roll angle
output.angular.y =
 ↪  pid_.roll.update(HeadingCommandHandle(*pose_input_).getError(*pose_),
 ↪  twist_->twist().angular.y, period);
```

Additionally the force and torque limit section in both `twist_controller_nodyn` and `pose_controller_nodyn` were commented out to not limit the force when the grounded controller is used.

B.3 Haptic feedback design

Project location

```
~/catkin_ws/src/rosfalcon/src
```

For generating the haptic feedback, the collision forces will be used from Gazebo. A plugin had to be written for this (section B.4.1). The main problem, which is discussed extensively in literature, is that physics engines have trouble with simulating collisions, especially between mesh objects [8], [9]. Results are often not realistic and generate unwanted behaviour. Each physics engine is optimized towards and written for a certain goal, e.g. speed, accuracy, realism. ODE is definitely an overall winner when comparing speed and accuracy but this has a trade-off where the simulation creates artefacts. One of the reasons I switched from Gazebo 2 to Gazebo 5 is to try out different physics engines. Gazebo 5 supposedly not only supports ODE but also Bullet, DART and Simbody. But unfortunately I couldn't get any of the additional (experimental) physics engines working. The newer versions of Gazebo (currently version 7, December 2015) are likely to have significant improvements with respect to these physics engines.

So I settled for ODE and started tweaking the simulation settings and model characteristics. A first look at the interaction forces from Gazebo was not pretty (Figure B.6). Two characteristics stand out here: the noise/jitter and the high initial contact forces.

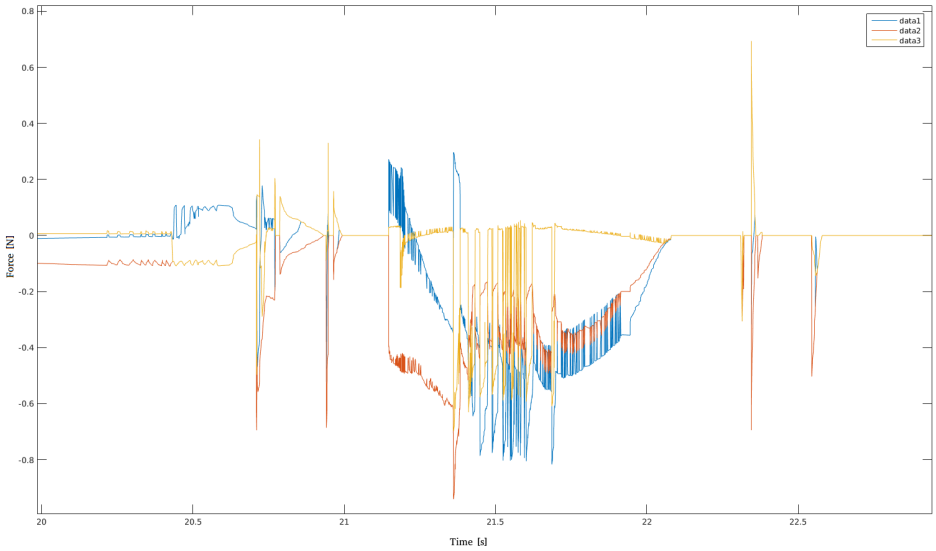
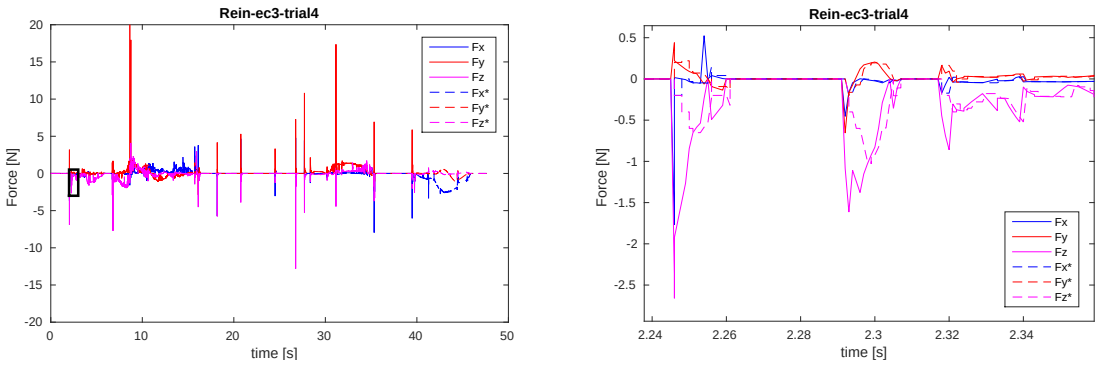


Figure B.6: A snapshot from the force levels (x, y, z) taken from `rqt_plot` where forces are plotted live during on of the first simulations.

To attempt to smooth the force signals, I first tried changing the object parameters, namely the stiffness and the damping. Gazebo's default stiffness and damping values are: $k = 1 \times 10^{12}\,\mathrm{N\,m^{-1}}$ and no damping. By using a reduced object stiffness of $k = 1 \times 10^{5}\,\mathrm{N\,m^{-1}}$ and a damping of $b = 1\,\mathrm{N\,s\,m^{-1}}$, the interaction was already a lot more stable. These values were chosen experimentally by performing interaction between the UAV and objects until it became visible that the UAV was penetrating into the object (too small stiffness). It is of course questionable how realistic these values are compared to an actual safety switch cover and switch. Future research should focus more on realistic interaction with improved physics engines as opposed to the stable interaction I tried to realise.

The second attempt to smooth the signals was to prevent large changes in forces during interaction (jitter). One common approach is to use a filter to smooth the signal. This initially didn't seem like a good idea to me because the force information from Gazebo is only updated at 1000 Hz and filtering this signal would likely result in a "spongy" or delayed haptic feedback. Another approach is to use a rate limiter to limit the increase/decrease per iteration step which is fast and does exactly what we want: limit fast changing forces (Figure B.7). What is clear from the figures below is that both the high initial contact forces as well as some of the jitter is reduced to a much smoother force signal.



(a) Forces from a whole trial

(b) Detailed forces from black boxed section in Figure B.7a

Figure B.7: Both unfiltered interaction force straight from Gazebo and the resulting "rate-limited" force. The left (Figure B.6) shows a complete trial and the right (Figure B.7b) shows a detailed view for a smaller time duration marked with a black box in the left figure.

The actual rate at which the force is limited per time step was chosen experimentally to be 0.2. A very low maximum rate will make no real difference as it will only remove very very big differences. A

very high maximum rate will induce the effect of a slow force increase that is too slow and linear as opposed to "instantaneous". This is bounded by the limit of the human force control bandwidth ($\sim$ 2 Hz). To make sure the rate limiter did not transform the force cues too much, a step response was used as input in both a rate limiter and a 2nd order filter with approximately the same ramp and then compared (Figure B.8).

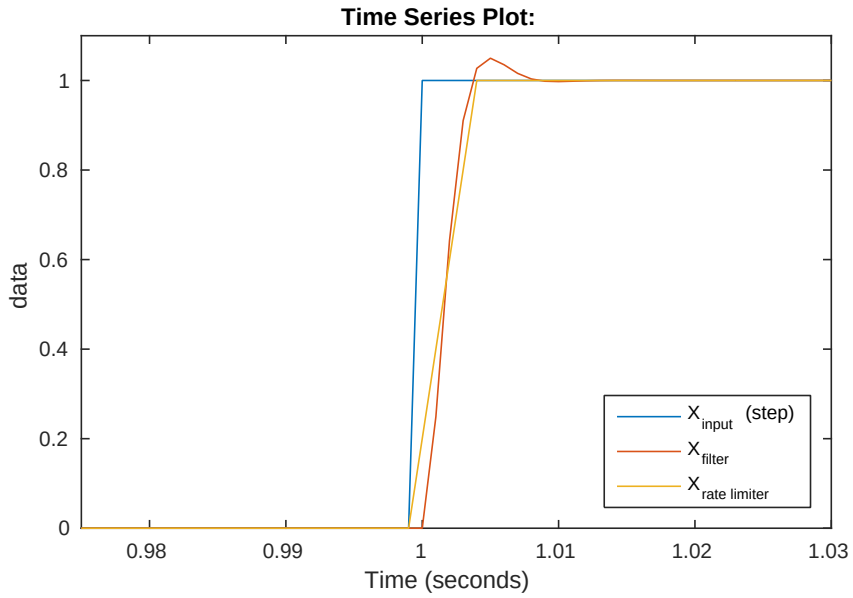


Figure B.8: Comparing the effect of a rate limiter to that of a 2nd order filter

In retrospect, I should have experimented more with the haptic feedback to research more on how others designed the force queues and how I could implement this. Unfortunately though, because the haptic feedback was only one independent variable that I was changing, it wasn't the only thing I could focus on. I would have liked to experiment more with the different rate limiters but also with filtering the Gazebo forces. ROS has standardized filters for processing data which would likely not even be that difficult to implement. I do however expect that conditions the forces from ODE or another physics engines is likely to be a study on its own.

B.4 Gazebo plugins

Project location

`~/catkin_ws/src/quad_world_plugins`

All the plugins created to interact with the virtual world in Gazebo are listed in this section and located in the folder above.

B.4.1 Contact force plugin

Project location

`~/catkin_ws/src/quad_world_plugins/ContactPlugin.cpp`
`~/catkin_ws/src/hector_quadrotor/hector_quadrotor_description/urdf/quadrotor_base.urdf.xacro`

A model plugin was created to measure the contact forces on the quadcopter [10]. The plugin can then be connected to the quadcopter model like so:

```
<!-- contact sensor -->
<gazebo reference="base_link">
```

```
    <sensor name="quadrotor_sensor" type="contact">
      <always_on>true</always_on>
      <update_rate>1000</update_rate>
      <contact>
        <collision>base_link_collision</collision>
      </contact>
      <plugin name="quadrotor_bumper" filename="libcontact.so">
        <bumperTopicName>quadrotor_bumper</bumperTopicName>
        <frameName>default</frameName>
      </plugin>
    </sensor>
  </gazebo>
```

The plugin works as follows, on every Gazebo update this routine is run:

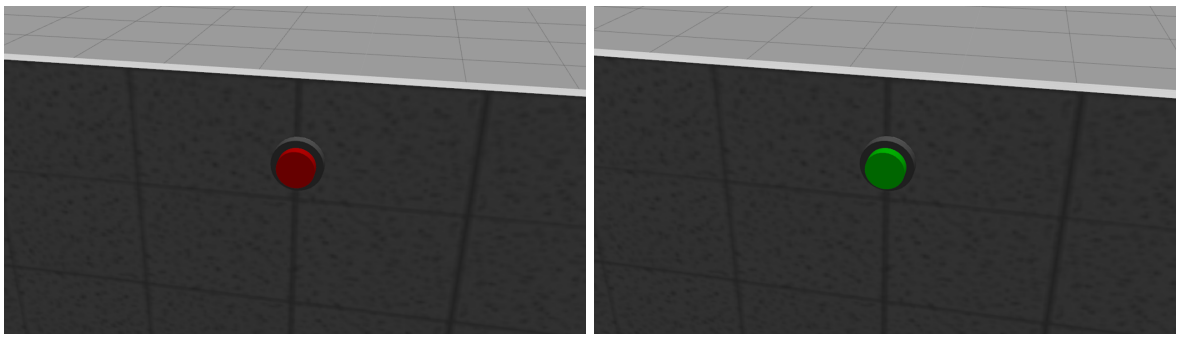
1. Get contacts from connected model
2. Get position of UAV from `/ground_truth/state` (to see if airborne)
3. If not airborne ($z < 1.12$ m), forces are 0 (forces are jittery if landed on platform)
4. If airborne, get forces from contact pointer
5. Publish `msgForce` to `/forces` topic

B.4.2 Indicator light plugin

Project location

`~/catkin_ws/src/quad_world_plugins/indicatorLightPlugin.cpp`
`~/.gazebo/indicator_light`

A model plugin was created to access the states of various models in a world [10]. The model is a floating indicator light that has a very flat shaped token that has a red and a green side B.9). The model will flip 180° to give the subject the indication that the task is complete.



(a) Task not yet completed (b) Task completed

Figure B.9: The two status possibilities for the safety switch model

The plugin can then be connected to the loaded model like so:

```
<plugin name="indicator_light_plugin" filename="libindicator_light_plugin.so"/>
```

The plugin works as follows, on every Gazebo update this routine is run:

1. Get safety switch cover angle from the `world_cover_joint` and the lever angle from the `world_lever_joint` inside the `safety_switch` model
2. If cover < 2 rad (fully open) then change cover status to 1 angle lever angle $< \frac{\pi}{3}$ rad (fully flipped) change task status to 1

3. If the task status is 1 then flip the indicator light model, which has a red front and a green back
4. Publish cover status to `/cover_status` and task status to `/task_status` which includes the current angles and a boolean status

B.4.3 Dirty wall plugin

Project location

`~/catkin_ws/src/quad_world_plugins/dirtyWallPlugin.cpp`

In this model plugin, the changing dirty wall surface was implemented to allow change of the dirty wall model. The dirty wall task is further explained in section B.1.2.

The plugin works as follows, on every Gazebo update this routine is run:

1. Get the current position of the quadcopter from `/ground_truth/state` and replace the old x and y positions and save the new x and y positions
2. Get the y-force acting on the dirty element (force perpendicular to the surface) and use the Archard equation and the defined parameters to calculate the total volume wear which is implemented to the number $0 \leq r \leq 1$
3. Change the color (both ambient and diffuse) of the dirty element where the RGB color of the element is set to vary between black (000) and red (100) by setting RGB = (r00)

B.4.4 Gazebo task widget

Project location

`~/catkin_ws/src/quad_world_plugins/GUITaskWidget.cpp`

I needed some kind of indication of how far the cover and the lever were open; not only for the subjects to know if the task was complete but also during testing to see how far it actually was open. The goal of this plugin was to give some kind of visual feedback of the task status in the Gazebo interface. I created a widget that shows the task completion in a percentage (Figure B.10).

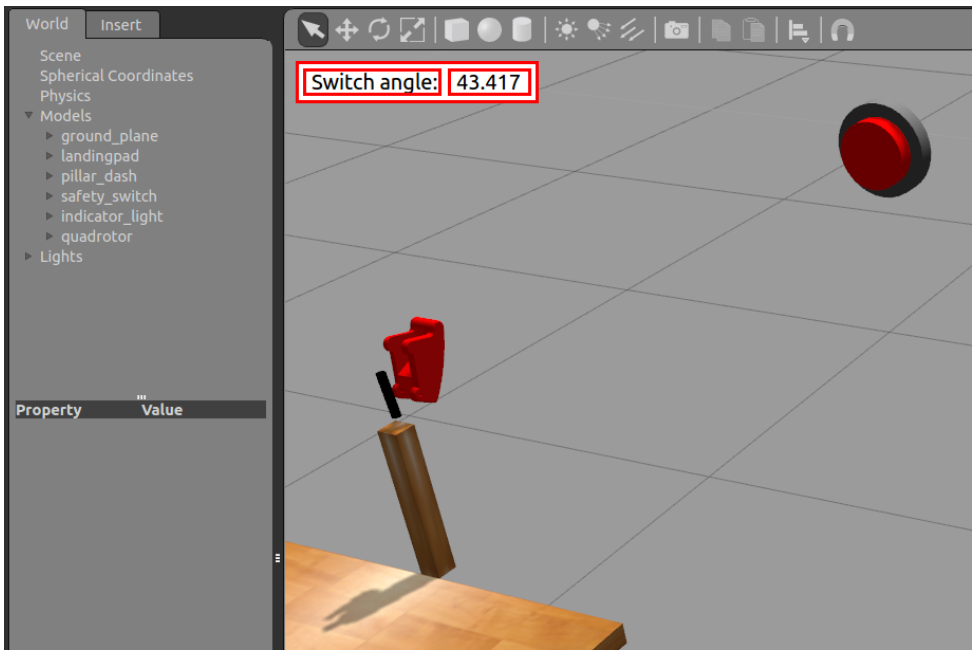


Figure B.10: A Gazebo widget that was designed to show the task completion of the switch in a percentage. The widget turns green when the switch angle has reached 100 % and thus the task is complete.

The plugin latches on to joint angle of the joint model named `world_lever_joint` and converts this into a percentage. You can use the widget in any world by adding the following code into the world section of a world file:

```
<plugin name="taskwidget" filename="libgui_task_widget.so"/>
```

Later I decided to implement an indicator light as explained in section B.4.2 as additional visual feedback for the subject and thus this widget was unused during the experiment. Nevertheless it could be useful for other purposes in future experiments.

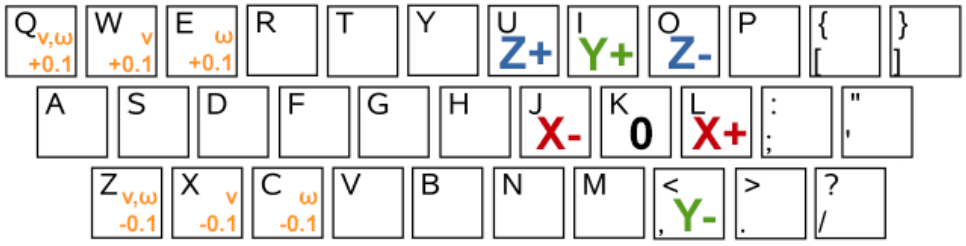
B.5 ROS nodes

B.5.1 Keyboard control node

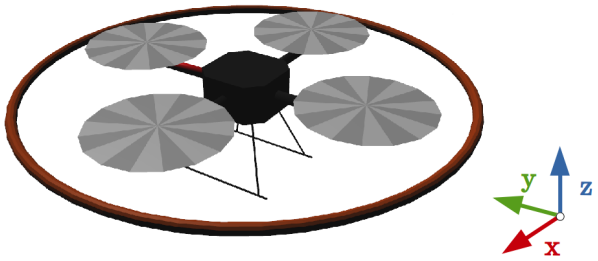
Project location

`~/catkin_ws/src/teleop_twist_keyboard`

The `hector_quadrotor` package includes nodes to control the quadcopter but only with a gamepad. Especially in the beginning and during testing, it can be useful to control the quadcopter simply with your keyboard instead of having to have an interface with you. Because the quadcopter subscribes to the ROS topic `/cmd_vel`, implementing this was easy. I altered the ROS package `teleop_twist_keyboard` to not only give velocity commands in for x and y but also for z. It works with the following keys:



(a) Keyboard input keys



(b) UAV axis definitions for reference

Figure B.11: How to control the UAV with a keyboard using the keyboard control node

The throttle is handled by the E, C, R, V keys add and subtract 0.1 (m/s or rad/s) from either the commanded linear or angular velocity. Additionally the Q, Z keys add and subtract 0.1 (m/s and rad/s) to both the command linear and angular velocity. The U, I, O, J, K, L, , keys set the direction in which you want to add the throttle to.

B.5.2 Falcon control nodes

Project location

```
~/catkin_ws/src/rosfalcon/src
~/catkin_ws/src/rosfalcon/src/joy2cmdpose.cpp
~/catkin_ws/src/rosfalcon/src/joy2cmdpose_HF.cpp
```

The rosfalcon is based on a older package called `cyphy_rosfalcon` which was developed by the Queensland University of Technology in 2010 and relies on [11]. It relies on `libnifalcon` which is explained in the Quick start guide in chapter A. Besides the fact that the package was no longer available to download, deprecated on the ROS site and not catkinized, it did provide a useful starting point for the node I created. After some emailing I received an archive from a current member of the Robotics lab at the Queensland University of Technology, as the creator was no longer employed at the University. I catkinized it which made it compile together with all the other ROS packages in my workspace and further changed it to fit my needs. I will go through the node step by step:

1. First the node does a whole bunch of stuff to initialize the falcon, it requires the libraries from `libnifalcon` for this. This was copied from an example node located in the src folder
2. To initialize successfully, it requires you to push the falcon all the way forward, then all the way back, and then all the way forward again. This records the hardware stops of the x-axis and sets them
3. Next, as long as the ROS node is running, it keeps looping all the subsequent items at 1000 Hz
4. During the first run through this loop, it homes the falcon in the middle of the workspace $(0, 0, 0)$ with a spring and waits for the operator to press the center button on the falcon to start the trial. Once the operator presses the button, the Falcon is released and can be moved around. The press of the button allows for the start time of each subject to be the same
5. The encoder positions and buttons statuses are read
6. The x axis is not centered (fully out is 0) so 0.124 m is subtracted to center the x-axis
7. The raw joystick positions are published to `/falconJoy`
8. The Falcon position is scaled to the Gazebo real world with a factor (default: $\times 4$)
9. The x and y axis are flipped to make them match Gazebo
10. A slight offset is added to the y-axis where we want the UAV to spawn a little further back, 0.05 m to be exact. Also for the z axis, 1.2 m is added because the middle of the workspace should correspond with the landing pad that the UAV is spawned onto
11. And finally, if the haptic feedback version of the node is running (`joy2cmdpose_HF`) then the forces are read from the topic `/forces` and published to the Falcon

B.6 Data acquisition

Project location

```
~/catkin_ws/bags
```

To compute metrics, data must somehow get from Gazebo and ROS to a program to further analyse it. Since I am proficient with Matlab, this was a logical choice for me.

B.6.1 ROS to MATLAB

Project location

```
~/catkin_ws/bags
~/catkin_ws/bags/bag2mat.sh
```

As far as I know there are two options to get information from ROS to Matlab. The first is through the Robotics System Toolbox which was introduced in Matlab version 2014a for the first time [12]. Though as with most toolboxes, this one is not always part of a standard Matlab package. The nice thing about this package is that it directly connects to ROS nodes allowing messages to be sent and received from inside Matlab. After the connection is made, it is as easy as setting Matlab to save messages from a certain node to a time series in the workspace. I tested this way of data acquisition and found that it was not only quite slow but also quite inefficient as it required me to run Matlab simultaneously with the simulation. The second and more efficient way of saving data is through ROS's built in recording and playback tools: `rosbag`. Except now the data is stored as a `.bag` file instead of a `.mat` file. Fortunately there are a number of ways to do this conversion of which using a python file called `bag2mat.py` was the fastest [13]. Since this python file only converts single files, I created a bash script to copy the recorded files from the default saving folder of ROS (`~/.ros`) and run it multiple times for all the `.bag` files in a folder. This is the script:

```
#!/bin/bash

echo "Copying all .bag files from .ros folder"
mv ~/.ros/*.bag ~/simulation/bags/exp

cd ~/simulation/bags/

echo "Creating file list of bag files"
ls exp/*.bag > baglist.txt

for x in $(cat baglist.txt)
do
  y="${x%.bag}.mat"
  if [ -f "$y" ]; then
    echo $y "Already exists"
  else
    python bag2mat.py $x $y
  fi
done

rm baglist.txt
```

B.6.2 MATLAB to results

Project location

```
~/catkin_ws/bags
```

Once the recorded messages from Matlab are stored in `.mat` files, importing them into Matlab is a piece of cake. I wrote a script called `expresults.m` which imports all the data into a structure, generates metrics and does a statistical analysis on all the data. You can use it to run on all my results from either the `test` folder, the `pilot` folder or the `exp` folder which contains all the mat-files of all the trials performed for testing, the pilot and the actual experiment.

Additionally, once all these results are loaded and the metrics are generated, you can use the script called `plotTrial.m` to plot individual trials. Say you want to plot the first trial of Anne from experimental condition 3 and you want to see the filtered resultant force that was exerted during the 4th trial, then you can do so by executing:

```
plotTrial('Anne','ec3','trial4','filtforceres')
```

The final argument for this function can be any of the following: `input`, `filtinput`, `traj`, `force`, `filtforce`, `filtforceres`, `velocity` or `all`.

Appendix C

Experimental setup and protocol

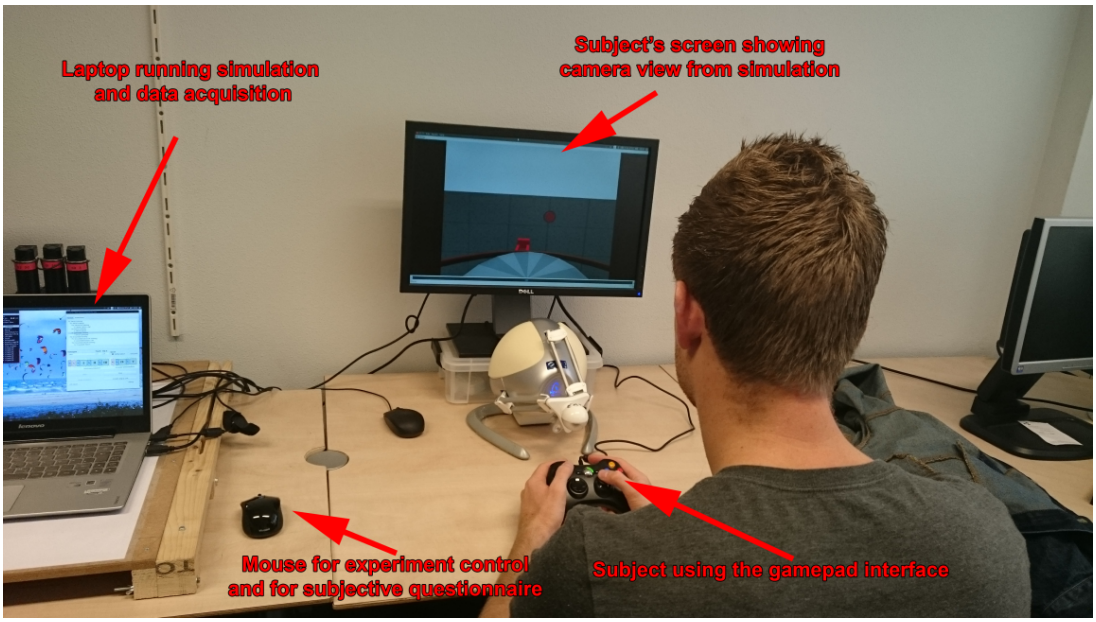
This chapter discusses the experimental setup, experimental protocol, experiment instruction and the informed consent used for this human factors experiment.

C.1 Setup

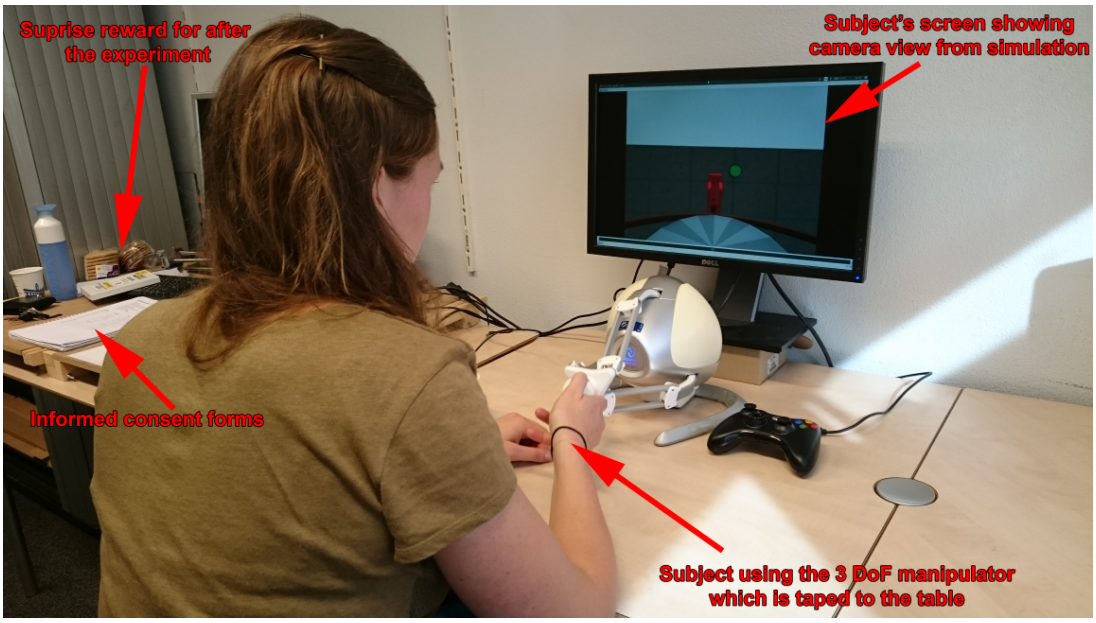
The setup used for this experiment is relatively simple and consists of (Figure C.1):

- Relatively powerful laptop (i7 4th generation, 8 GB RAM, GT730M 2 GB) running:
  - Ubuntu 14.04 (trusty)
  - Gazebo 5.1
  - ROS Indigo
  - My experiment UAV telemanipulation GUI (See section A.2.2)
- Wireless mouse
- External widescreen monitor (23")
- Novint Falcon connected with USB
- Microsoft Xbox 360 Wired controller connected with USB

The Novint Falcon had to be taped to the table to prevent undesired sliding due to excessive forces by or the subject or the haptic feedback.



(a) Subject performing trial viewed from the left, using the gamepad

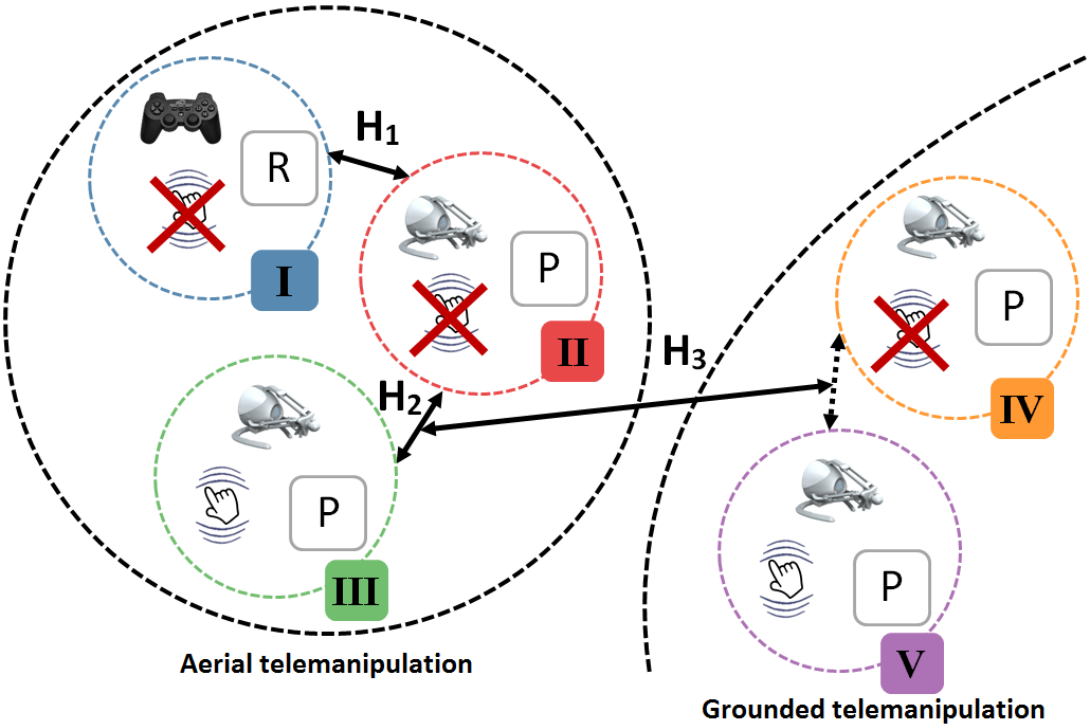


(b) Subject performing trial viewed from the right, using the 3 DoF manipulator

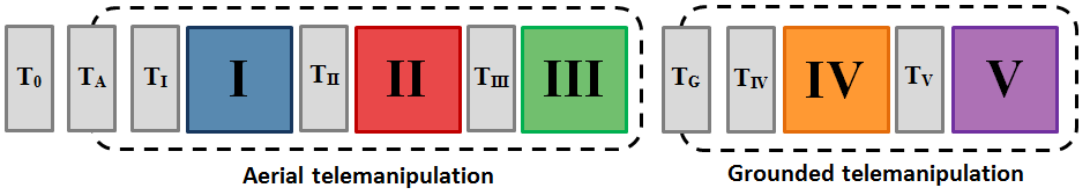
Figure C.1: Experimental Setup with all the components

C.2 Protocol

A total of five experimental conditions were testing during this study. Initially only three conditions were planned: the baseline (gamepad with rate control), the 3-DoF manipulator with position control and the same manipulator with haptic feedback. At a certain point we became aware that, since we were the first to conduct such human-in-the-loop aerial telemanipulation experiments, we were skipping a step. We were planning to research aerial telemanipulation but had no clue as to how it compares or differs with classical grounded telemanipulation. We expected telemanipulation with an aerial slave to (logically) be harder but exactly how much harder? Following on that, because we expected it to be harder, we expected haptic feedback to have more influence. Since the task would be less effective with an aerial slave, the relative benefit of haptic feedback was expected to be higher than with a grounded slave. Thus two extra experimental conditions were added making a total of five (Figure C.2).



(a) The five experimental conditions used in this research.



(b) Experimental protocol

Figure C.2: The experimental conditions shown in relation to other literature (Figure C.2a) and as experimental protocol (Figure C.2b). The five experimental conditions indicated as I, II, III, IV and V with their respective color which are used throughout this paper. The three hypothesis (H_1, H_2 and H_3) and their respective relationships are shown as arrows. The icons represent the interface type (gamepad or manipulator), whether haptic feedback is present, and the type of controller: rate (R) or position (P).

To insure a relatively equal skill level of each subject, each experimental condition was proceeded by a training period to get the subject accustomed to the conditions. These training sessions were repeated at minimum two times and at most five times. Every subject has a different learning curve rate and gaming experience which result in a variable number of training sessions per subject. Subjects would often flip the UAV the very first time, despite that they were instructed how to prevent this. During the training I would watch the subjects and give additional tips to help them learn faster and after a few sessions, we would collectively agree that they understood the task and were ready for the experimental conditions.

During these sessions, the same safety switch task was presented to the subjects except no data was recorded (T_I through T_{IV}). Besides these these training sessions, a training session to introduce the slave dynamics type (T_A through T_G) and the simulation itself (T_0). During both these last training sessions a training world was used where the task was simply to fly around and get accustomed with the dynamics and the simulation view itself. Additionally the very first training session (T_0) presented two views to the operator: the normal first person (camera) view and a third person view from the top (Figure C.3). This was only during this specific training condition to give the subjects a feeling for the room they would be operating in.

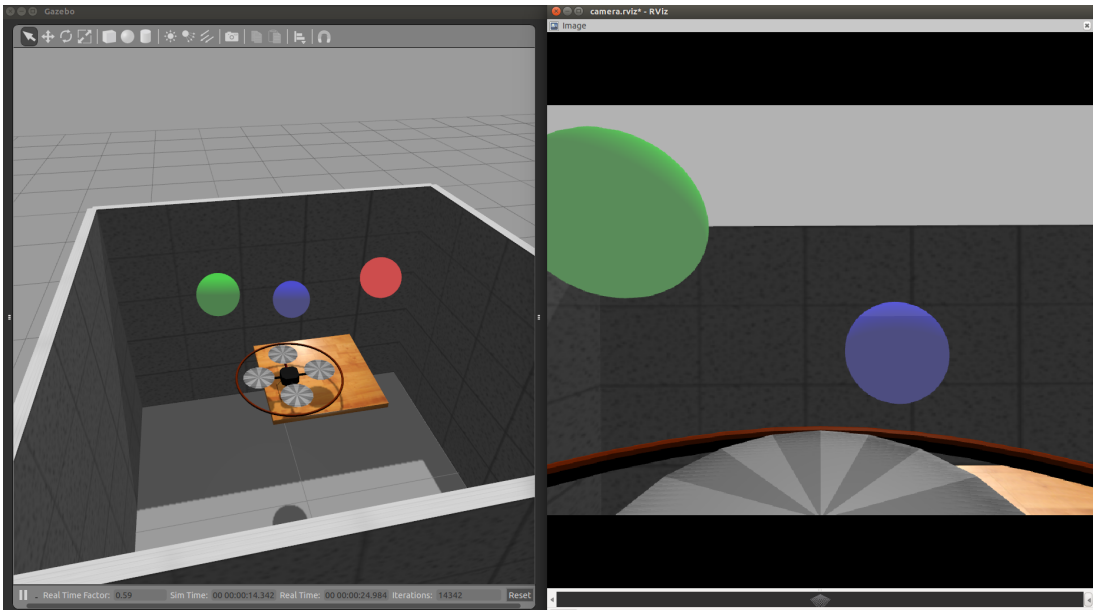


Figure C.3: The very first training session (T_0) which had both the first person (camera) view and a third person view of the simulation environment.

Due to these training effects and the simple fast that subjects get accustomed to certain interface behaviour, the experimental conditions had to randomized (Figure C.4). This was the conditions would be presented in a different order for each subject so that if training effects would cancel each other out over the whole experiment. The experiment would start with either aerial or grounded slave dynamics and then continue with the internally randomized conditions within this group.

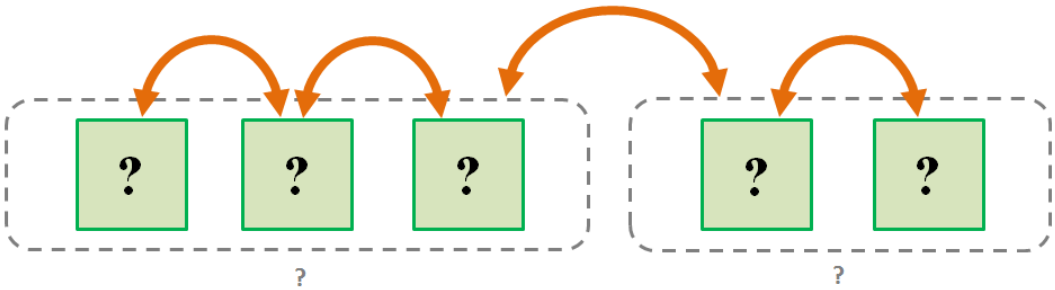


Figure C.4: The experimental conditions and their respective training sessions were randomized. The experiment always started with the setup training (T_0) and then proceeded with either the aerial or the grounded slave conditions. The internal order of either the aerial or the grounded slave conditions was also randomized.

C.3 Instructions

It is important that the subjects in this experiment all follow a certain instructions to prevent too much variation between subjects. We want to relate the task to performing tasks in reality as much as possible. Therefore the instruction are such that the subjects should do their very best to perform the tasks as **fast** and as **careful** as possible. Fast because in disaster areas, time is of primary concern and careful because once you get a UAV into a disaster area, you don't want to damage both the fragile environment even more. In addition, you don't want to damage the UAV either so tasks must be performed as if you only have one UAV. A crashed UAV in a disaster site is just as useless as a ground robot that is stuck between a pile of rubble.

This human factors study followed this structure: Participants are invited to sign up for a time slot, receive written instructions, sign an informed consent form and then perform the experimental conditions. I started writing the written instruction but they soon became so long and complex. I wondered how long

it would take to make instructions that would keeps subjects interested and more importantly inform them well enough to perform the tasks. I decided it would be more fun and better to inform subjects what they could expect with a instruction video (Figure C.5). This video does all the same things that written instructions would only the instructions are more visual, include demonstrations and are thus hopefully clearer.

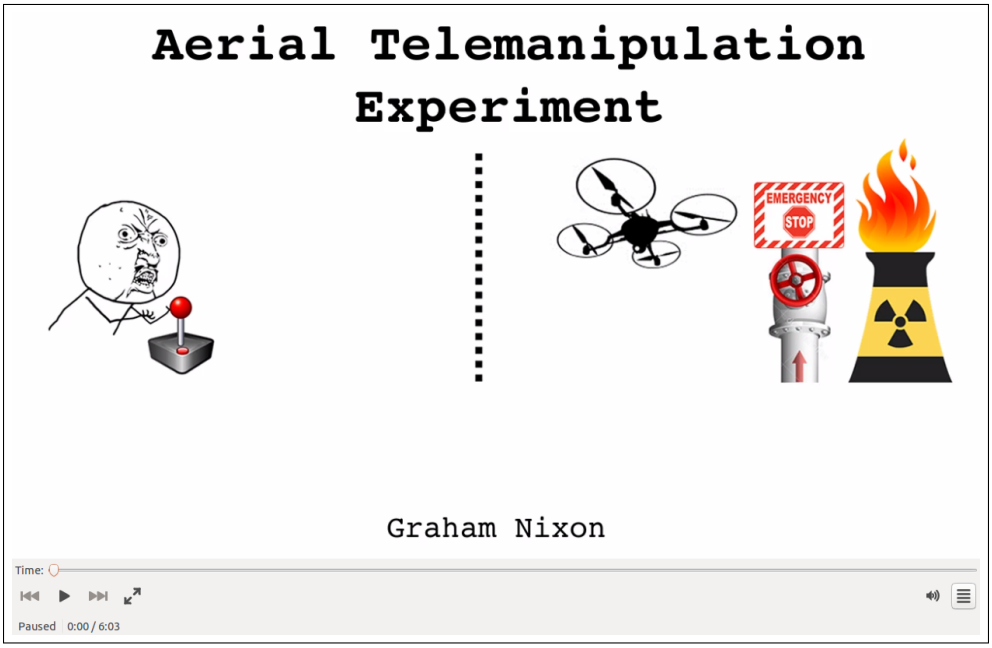


Figure C.5: Instruction video explaining the background, the interfaces, the experimental conditions and the task instructions of the experiment.

The video is only accessible through the link below and is not publicly listed on Youtube.

⇒ `https://youtu.be/FwKoXvl10Ac` ⇐

C.4 Informed consent

A standard consent form was used which participants had to sign before partaking in the experiment (Section C.4.1). This form together with the experimental protocol was approved by the Delft University of Technology Human Research Ethics Committee (HREC).

Informed Consent Form – Aerial telemanipulation Interface Design

Dear participant,

You have been asked to participate in a study on aerial telemanipulation. The research is conducted by Graham Nixon, under supervision of Jeroen Wildenbeest and David Abbink. In this study the effect of interface designs on task execution with a Unmanned Aerial Vehicle (UAV or drone) is studied. More specifically, this study aims at increasing the understanding how interface design influences performance during aerial telemanipulation.

You will be requested to perform a number of tasks in a virtual reality simulation with 5 different interface designs. Your goal is to perform the task as fast and as careful as possible. For each of the five systems, you'll be performing the task 5 times, making a total of 25 trials. The total experiment is expected to last about 1 ½ hour.

Your answers, as well as position data of the devices and the virtual UAV are recorded. These recordings are used anonymously. Personal data is not available to persons other than the researchers. Participation in this study is voluntary. If you feel any form of discomfort during the experiment, please inform the experimental leader. You are free to withdraw from the experiment at any time. For questions after the study, please contact Graham Nixon (g.a.n.nixon@student.tudelft.nl).

I, the undersigned, declare to have read and understood the information about this experiment, the use of data and to consent in the experiment.

Name: _____

Location: _____

Date: ____ / ____ / _____

Signature: _____

Appendix D

Measuring effectiveness

This appendix explains how these co called metrics for the effectiveness of the interface designs were chosen and calculated. Two types of metrics are discussed: Objective metrics derived from signals that were measured directly from the simulation and subjective metrics which were calculated from a questionnaire which was presented to the subjects after each experimental condition.

D.1 Objective metrics

The following objective metrics were considered as indication of the effectiveness of the interface designs. This section motivates their use, shows how they are calculated and how suitable they turned out for measuring reflecting what we are trying to measure.

D.1.1 Completion time

Variable

`completionTime`

Motivation

In a disaster scenario, there are two primary concerns: speed and caution. Speed because every second counts and caution because you don't want to damage the environment or the UAV in the process. The time to complete the task is a very logical and in our case the primary indication of how well the task is performed. A larger completion time indicates the task more difficult to perform due to the changes made to the interface design.

Calculation

Because both the simulation and the interface have to initialize, even though signals are already recorded, the completion time doesn't start until the subject pushes the button. On the gamepad the button ID = 1 and on the manipulator the button ID = 4. The timer stops at the moment when the switch is fully activated and the indicator light turns green. Completion time is independent of the cover angle even though the cover must practically be fully open to be able to reach the switch. It is calculated as follows:

$$\texttt{completionTime} = T_{\text{switched}} - T_{\text{button}} \tag{D.1}$$

Suitability

The metric is sensitive enough to reflect performance and will be used as primary metric for performance.

D.1.2 Trajectory distance

Variable

`TrajLength`

Motivation

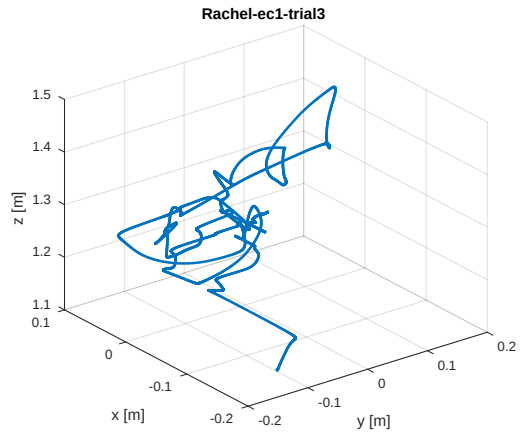
The distance that is covered by the UAV is another way of measuring the performance of the task. The longer the distance travelled the less effective the task was performed.

Calculation

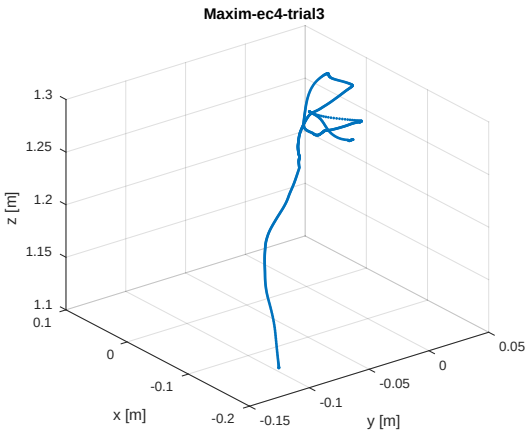
The trajectory distance is calculated as follows:

$$\texttt{trajLength} = \sum_{i=2}^{b} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2} \tag{D.2}$$

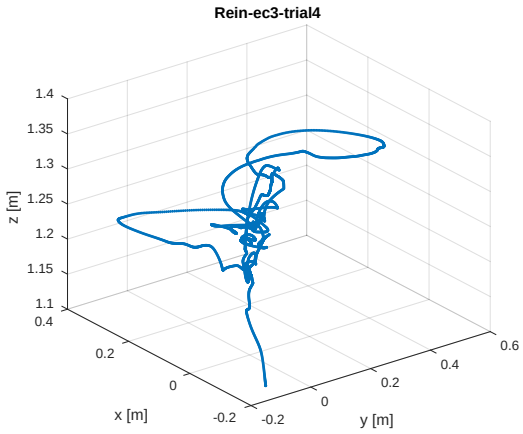
Where i is the current sample and b is the total amount of samples. Figure F.2 shows two examples of the trajectory of the UAV during two different trials.



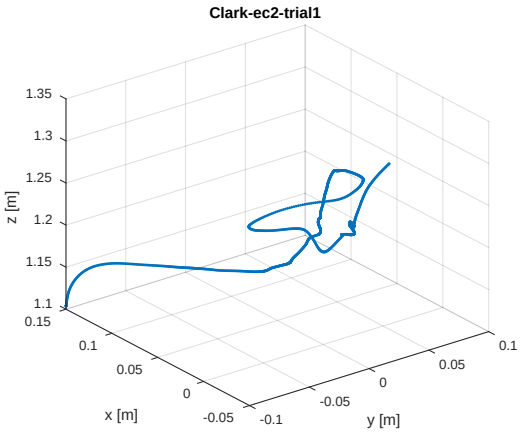
(a) Trial with the gamepad (I) (3.03 m, poor performance)



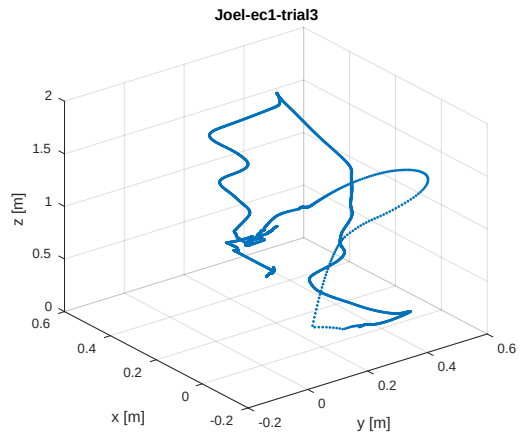
(b) Trial with the grounded manipulator (IV) (0.55 m, high performance)



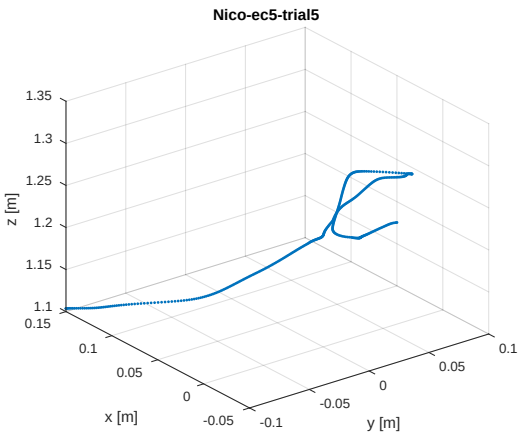
(c) Trial with the manipulator with haptic feedback (III) (4.38 m), poor performance)



(d) Trial with the manipulator (II) (0.72 m, high performance)



(e) Trial with the gamepad (6.43 m, poor performance)



(f) Trial with the grounded manipulator with haptic feedback (V) (0.54 m, high performance)

Figure D.1: Representative trials shown in 3D plots of the trajectory of the UAV during good and bad trials with in brackets the experimental condition and the respective trajectory length

Suitability

From the figures above it is clear that the shorter line (Figure D.1b) is more effective than the longer line with more variability (Figure D.1a). Though this metric is a good indication of how effective the operator

performed the task, it reflects the same as the completion time metric. Since the task instruction was to perform the task as fast and as careful as possible, completion time is better metric for performance.

D.1.3 Number of flips

Variable

`flip`

Motivation

Since the instruction is to perform the task as fast and as careful as possible, a good measure for this "carefulness" is the number of times that the UAV flipped due to faulty positioning and incorrect steering. More flips means the operator was more careless.

Calculation

To count the number of flips, a very simple approach was taken. If the trial was finished before the switch reached the final angle and completed the task, the trial was not loaded into the results any further and the number of flips was raised by 1. Another approach would be to look at the pitch or roll angle of the UAV (ϕ around x-axis and θ around y-axis) and see if they were ever larger than π. The former method was chosen because is was computationally more efficient.

Suitability

Though this metric is very sensitive to operator "caution", only whole values are represented. Flips occurred solely during the first condition (gamepad with rate control) with exception of 1 flip in a trial during the second condition (manipulator with position control). Furthermore the average flips in the first condition was 0.3429 and the standard deviation 0.2251 making the amount of flips so low that though the data was normally distributed, the power to measure significance of the number of flips is simply too low. For this reason, in the end, this metric was not used for measuring caution.

D.1.4 Standard deviation of the input

Variable

`stdinput`

Motivation

The standard deviation of the master device is a well known metric for the amount of variability of the operator input. It is often used as a metric for physical effort: A higher standard deviation thus reflects more effort.

Calculation

The input signals is is first filtered with a 3rd order 4 Hz butterworth filter and then the function `std` from Matlab was used to calculate the metric.

$$\texttt{stdinput} = \texttt{std}(x_{\text{input},i}) \tag{D.3}$$

Suitability

Because of the dissimilar master devices but also their respective controllers used in this study, the standard deviation does not reflect effort very well. A rate controller requires the input device to have a non-zero position to reach a new position whereas a position controller does not. On the other hand, the zero position of a position controller does not represent anything meaningful besides a certain reference point. Since standard deviation measures the variability between the x-axis and the signal, this simply won't hold. Therefore the standard deviation of the input will not be used as metric for the effort.

D.1.5 Number of reversals

Variable

`revrate`

Motivation

The reversal rate or the number of reversals is, just like the standard deviation, a measure for how much activity is being put into the input device by the operator. By counting the amount of times the operator applies a correction and changes direction a value can be given for the effort, except now for mental effort. Subjects with more reversals are having more trouble to generate the correct input signal to perform the task well.

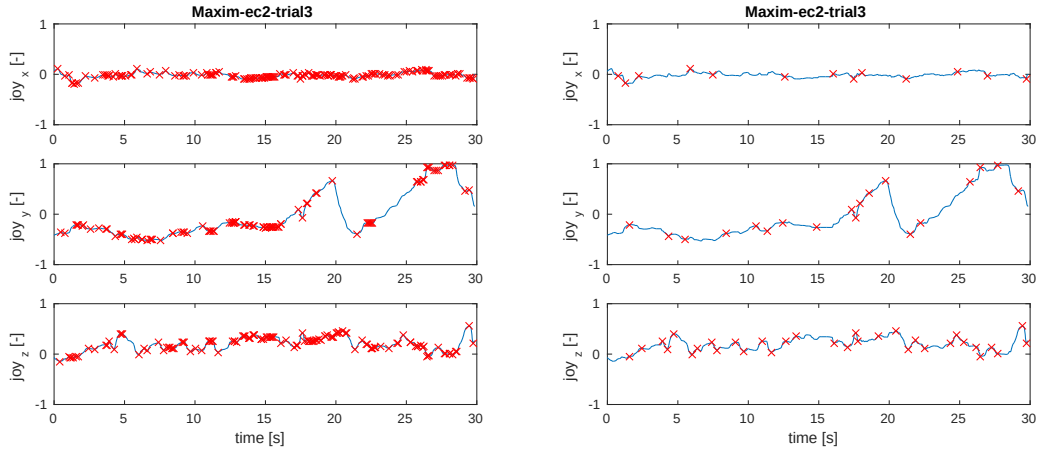
Calculation

First of all, the derivative of the steering input has to be 0, as follows:

$$\frac{\mathrm{d}x_{input,i}}{\mathrm{d}t} = 0 \tag{D.4}$$

Where $x_{input,i}$ is vector with the control inputs and i is the axis number representing the x, y or z-axis.

But if we only look at the zero crossings, unintended reversals are also counted (Figure D.2a). Not only very small changes in input are counted but also reversals that are really close to each other. Therefore we add a second condition that will determine if there has been a change in input larger than a certain threshold (c) has been exceeded since the last reversal where $c = 0.1$ which is 10 % of the maximum joystick deflection (Figure D.2b).



(a) Only checking for zero derivative

(b) Checking for zero derivative and last reversal

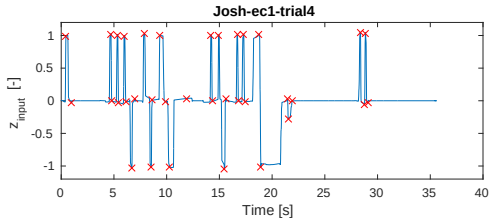
Figure D.2: Simple and more elaborate reversal count

Suitability

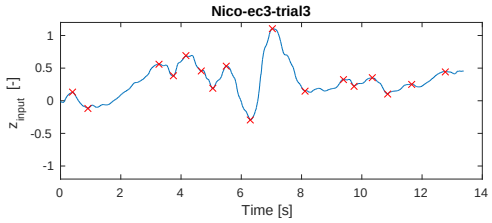
Just like when comparing standard deviations from different master devices and controllers, the same problems occur: the meaning of the absolute zero is not the same. Additionally, a rate controller requires at least one reversal to reach a certain reference position whereas a position controller does not. This means that the comparison of the number of reversals is not ideal between the gamepad and the manipulator. Nevertheless, it still does convey a measure for how much mental effort the subject is putting into the master device. Moreover, other metrics will likely also suffer from the dissimilar master devices therefore we will still use the number of reversals as the measure for effort.

We will only be looking at the reversals in the y and z plane because these are the only directions in which the operator is required to perform steering inputs. The UAV does start at an offset in x-direction but after the subject has corrected for this offset, corrections in the x-direction are not needed.

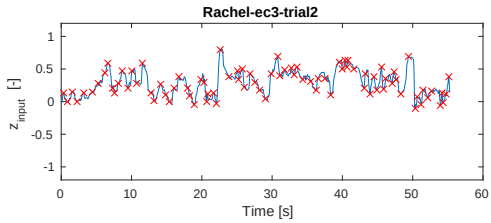
Below are a few representative examples with the number of reversals marked in each trial only in z-direction (Figure D.3). The z-direction is chosen because it is a relevant axis where reversals are needed to perform the task, as opposed to the x-axis.



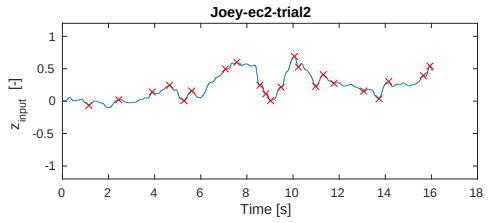
(a) Trial on the gamepad (I) with high number of reversals (high effort)



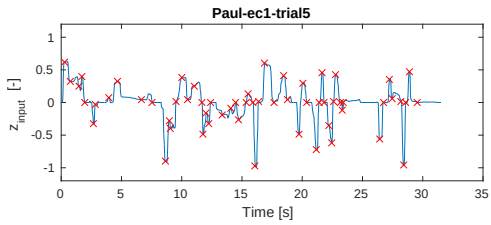
(b) Trial on the manipulator with haptic feedback (III) with low number of reversals (low effort)



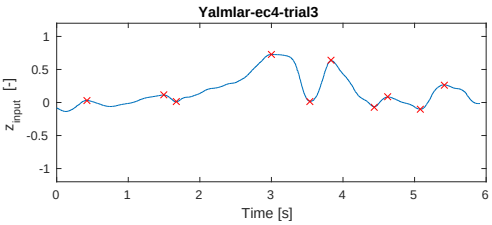
(c) Trial on the manipulator with haptic feedback (III) with high number of reversals (high effort)



(d) Trial on the manipulator (II) with low number of reversals (low effort)



(e) Trial on the gamepad (I) with high number of reversals (high effort)



(f) Trial on the grounded manipulator (IV) with low number of reversals (low effort)

Figure D.3: Representative trials with a high and low number of reversals only in the z-direction with in brackets the experimental condition.

D.1.6 Average force

Variable

`F_avg`

Motivation

The average force applied is a way to measure how effective a task is being performed. A low average force indicates that, on average, very little force is being applied which means it will take longer to perform the task. It could also be a measure of how cautious a task is being performed though the metric discussed next (maximum force) is a better measure for that.

Calculation

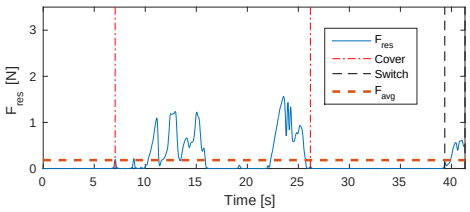
The resultant contact force is first filtered with a 3rd order 4 Hz butterworth filter and then the mean of this signal is taken from the moment the subjects pushes the start button until the task is complete.

$$\texttt{F\_avg} = \texttt{mean}(F_{\text{res,filtered}}) \tag{D.5}$$

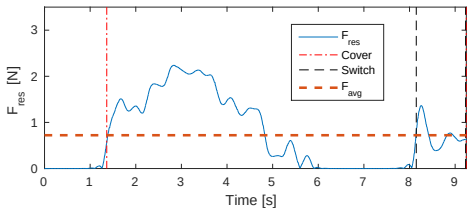
Suitability

Since the completion time will already be used as a primary indication of subject performance and the number of reversals for the subject effort, we are looking for a secondary metric that will show a different

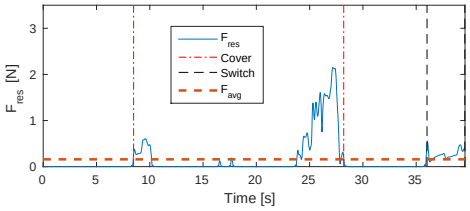
aspect of the subjects performance than the completion time. As can be seen in the figure below, the difference is clearly visible in contact force and average contact force between a good and a poor trial (Figure D.4). Therefore the average contact force will be used as secondary metric for performance during the analysis.



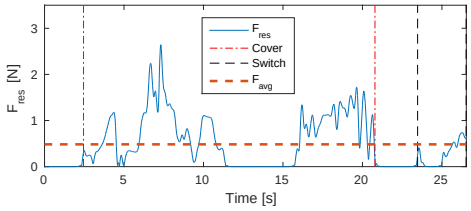
(a) Trial on the gamepad (I) with low average contact force (low performance)



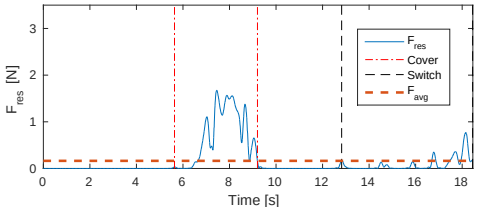
(b) Trial on the grounded manipulator with haptic feedback (IV) with high average contact force (high performance)



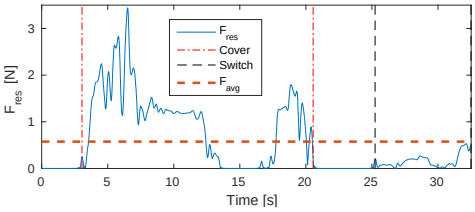
(c) Trial on the gamepad (I) with low average contact force (low performance)



(d) Trial on the manipulator (II) with high average contact force (high performance)



(e) Trial on the manipulator (II) with low average contact force (low performance)



(f) Trial on the manipulator (II) with high average contact force (high performance)

Figure D.4: Illustration of the average contact force metric for a trial with low and high average contact force.

D.1.7 Maximum force

Variable

`F_max`

Motivation

Besides the average contact force, the maximum contact force can also be calculated for each trial. This is more of a metric for how cautious the task is performed as opposed to how effective and could therefore be interesting to analyse the trade-off between speed and carefulness.

Calculation

The resultant contact force is first filtered with a 3rd order 4 Hz butterworth filter and then the maximum value of this signal is taken from the moment the subjects pushes the start button until the task is complete.

$$\texttt{F\_max} = \texttt{max}(F_{\text{res,filtered}}) \tag{D.6}$$

Suitability

For analysing the suitability of this metric the same figures as for the average force are representative for a good and bad trial (Figure D.4). What is clear here is that the maximum force is about the same for both a good and a bad trial. This is because this is most often the force that occurs when the cover is first moving and then hits the end stop. Due to this effect, this metric doesn't represent what we are looking for and thus won't be used for a measure of performance.

D.1.8 Maximum velocity

Variable

`V_max`

Motivation

The maximum velocity is, similar to the completion time, a way to capture how fast a subject is performing a trial.

Calculation

The maximum velocity is calculated by first finding the maximum velocity in all three direction: x, y and z and then taking the maximum of these three.

$$\texttt{V\_max} = \texttt{max}(V_{\max,x}, V_{\max,y}, V_{\max,z}) \tag{D.7}$$

$$V_{\max,i} = \texttt{max}(V_i) \tag{D.8}$$

Suitability

Because again, just like with the maximum force, the maximum velocity captures a specific event which occurs in just about every trial, it is less suitable for measuring how well a subject performs. The maximum velocity is directly bounded by the maximum velocity which the rate controller reaches with the gamepad during maximum joystick deflection. Similarly with the position controller on the manipulator, it is bounded by the controller parameters. Therefore it will not be used as a metric for analysis.

D.1.9 Number of contact transitions

Variable

`F_trans`

Motivation

During a trial, the number of times the UAV hits the cover or the switch is a good indication for how effective the subject is performing the task because ideally the subject flies towards the cover and opens it in one go. Similarly for the switch, the task is ideally performed in one try. Of course, due to various reasons, it usually takes a few tries before the right contact point is discovered.

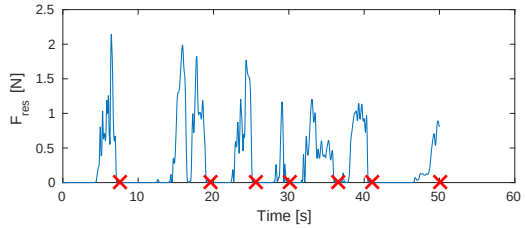
Calculation

For the number of contact transitions, the resultant force, filtered by a 3rd order, 4-Hz low-pass filter, is used to calculate the number of transitions. The number of contact transitions is subject to a number of thresholds: F_0, t_0 and F_{fall}.

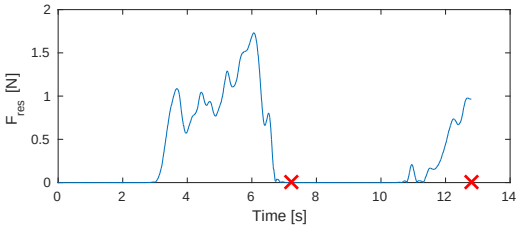
1. $F_0 = 1 \times 10^{-4}\,\text{N}$, the threshold which the force has to be below to count as a transition from high to low.
2. $t_0 = 0.25\,\text{s}$, the time threshold that the force has to be low to count as a transition.
3. $F_{\text{fall}} = 0.1\,\text{N}$, the threshold that the force has to be above while falling to count as a transition.

Suitability

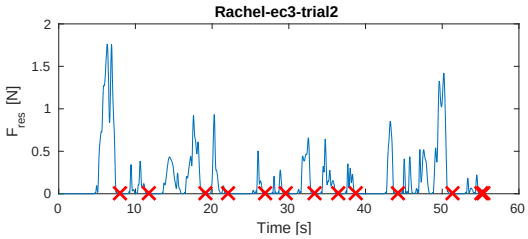
As can be seen in the poor and good trials below, the number of times the UAV bumps against the cover or the switch does reflect how well the subject is performing the task (Figure D.5). Completion time is also significantly lower which is used as a primary indication of performance. An issue is that the number of contact transitions is highly dependent on the thresholds chosen and can very significantly when these thresholds are changed, Additionally, the metric is similar to the number of reversals except the reversal rate is measured on the input device whereas the number of contact transitions is measured on the slave. Because of this similarity, it was chosen not to be used a secondary metric for performance because.



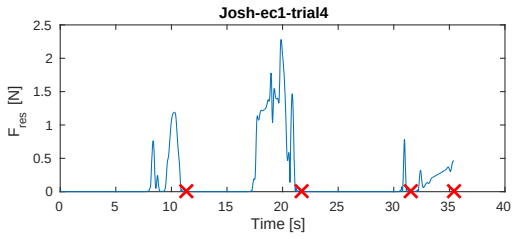
(a) Trial with the manipulator (II) with high number of contact transitions (low performance)



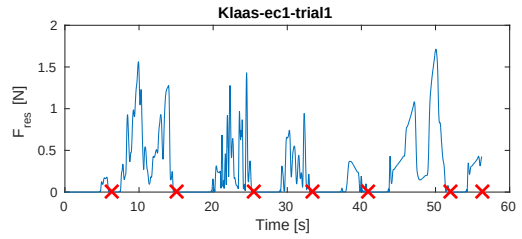
(b) Trial with the manipulator (II) with low number of contact transitions (high performance)



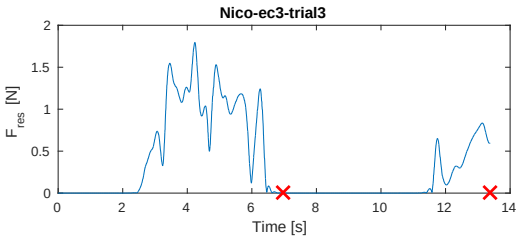
(c) Trial with the manipulator with haptic feedback (III) with high number of contact transitions (low performance)



(d) Trial with he manipulator with haptic feedback (III) with low number of contact transitions (high performance)



(e) Trial with the gamepad (I) with a high number of contact transitions (low performance)



(f) Trial with tthe manipulator with haptic feedback with low number of contact transitions (high performance)

Figure D.5: Representative trials showing the number contact transitions metric for trials with low and high amount of transitions. Contact transitions marked with red crosses.

D.1.10 Inefficient time

Variable

`ineffTime`

Motivation

Though completion time is the primary indication used for effectiveness of the trial, it has one main advantage: It not only captures the manipulation time (which is of primary interest) but also the free flying time. It seems useful to use a more specific measure for time where only the actual time that manipulation is taking place is considered. By calculating the time that the UAV is in contact with either the switch or the cover and also calculating the time that the switch or cover are moving, a

measure can be created for the so called ineffective time: The time that the UAV is manipulating the task, but without result.

Calculation

$$\texttt{ineffTime} = \texttt{contactTime} - \texttt{manipulationTime} \tag{D.9}$$

Suitability

This metric reflected performance very well because it captures exactly what we are interested in: How well the UAV is actually flipping the cover and the switch. But because this metric showed very similar results to the primary metric: completion time, it was chosen to not include a very similar metric as secondary performance metric. This metric was briefly considered as primary metric instead of completion time but in the end, the initial choice for completion time provides the most simple and salient metric. More complicated metrics may show higher significance but due to their complexity they also reflect less meaning.

D.2 Subjective metrics

Besides the aforementioned objective metrics, it is interesting to comment on how the subject experienced the interface designs in relation to each other. Subjective measures are especially useful when the objective measures turn out to show that there is no real difference. The subjective measures can then give more insight on what is happening inside the subjects head. The NASA TLX method is a often used comprehensive tool to measure mental, physical en temporal demands as well as performance and effort. But it comes at the cost of a lot of questions for the subject to answer. Additionally, it may provide us with more information then we are interested in. Another way of measuring the so called acceptance of a system is by using the Van Der Laan acceptance questionnaire. This method is a lot simpler and shorter and gives us exactly the information we meed: The subjective altitude towards the new interface design. Therefore this method will be used as subjective assessment of the subjects.

The Van Der Laan questionnaire is a procedure for measuring acceptance of new technology where a five point likert scale (-2 to 2) is assessed by scoring nine items [14]. These nine items can then be used to compute scores for both usefulness and satisfaction. Practical aspects of the system are reflected by the usefulness score whereas the satisfaction is a measure for the pleasantness of the system. Practical aspects of the system are reflected in the usefulness score, while the pleasantness is mirrored in the satisfying score.

My judgements of the (...) system are... (please tick a box on every line)			
1	useful	_ _ _ _ _	useless
2	pleasant	_ _ _ _ _	unpleasant
3	bad	_ _ _ _ _	good
4	nice	_ _ _ _ _	annoying
5	effective	_ _ _ _ _	superfluous
6	irritating	_ _ _ _ _	likeable
7	assisting	_ _ _ _ _	worthless
8	undesirable	_ _ _ _ _	desirable
9	raising alertness	_ _ _ _ _	sleep-inducing

Figure D.6: The 9 Van Der Laan questionnaire items

As a questionnaire tool, Google Forms was used because the subjects were already sitting behind a computer screen and it nicely orders the results in an Excel form. Below are two screenshots of the first page of the questionnaire (Figure D.7) and the 9 likert scale items that were presented per interface design (Figure D.8).

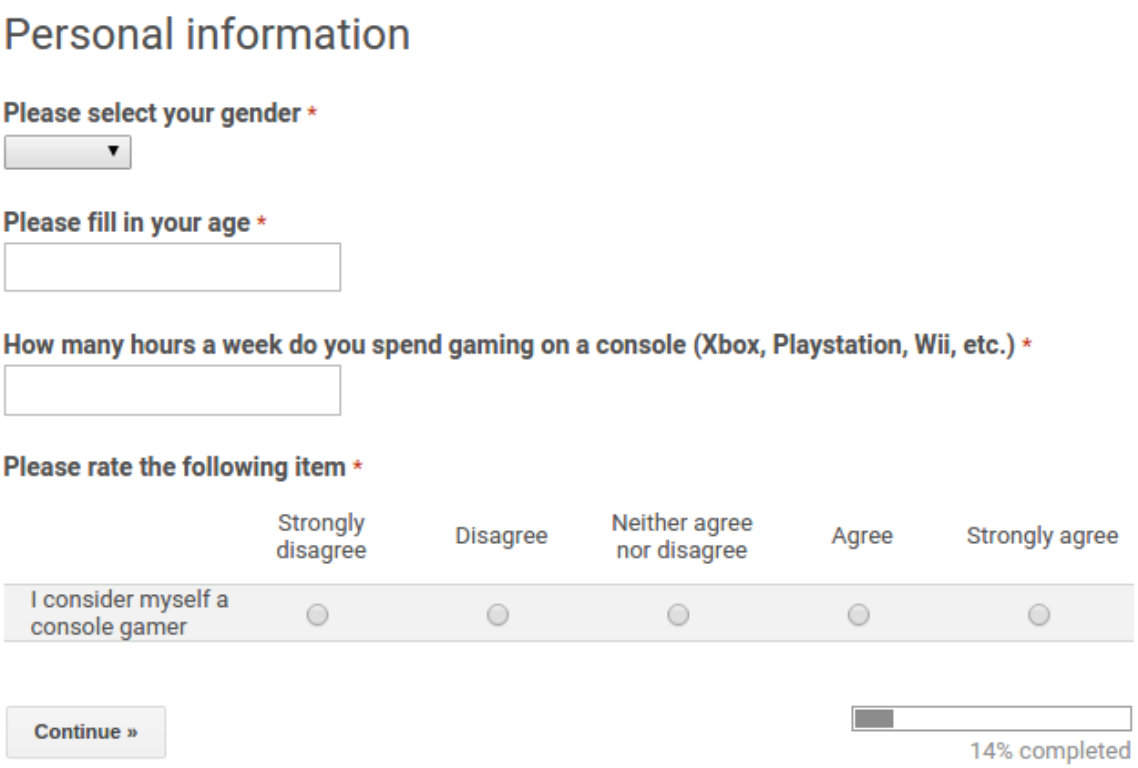
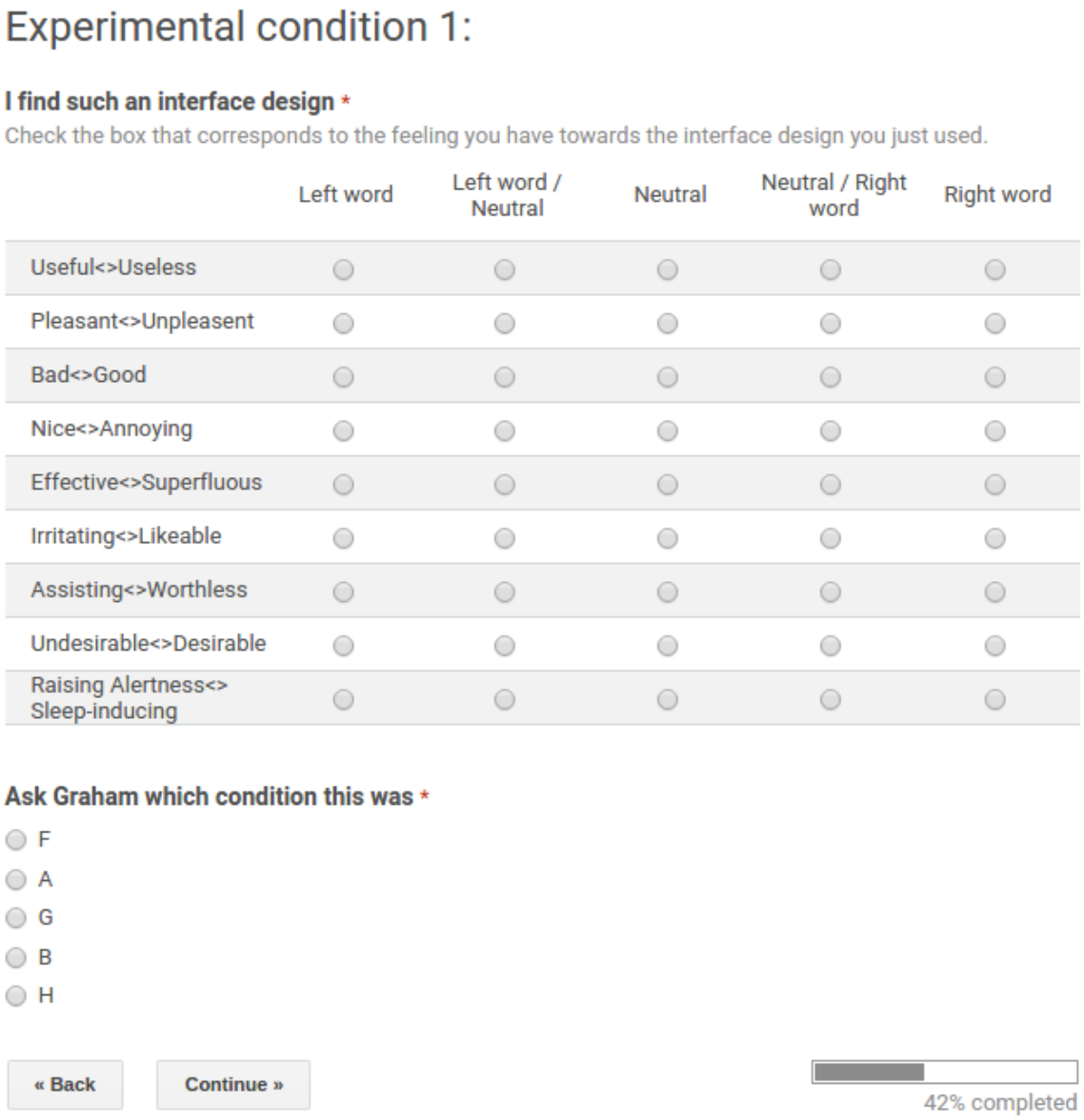


Figure D.7: Page 1 of the questionnaire showing personal and gaming experience questions



Experimental condition 1:

I find such an interface design *

Check the box that corresponds to the feeling you have towards the interface design you just used.

	Left word	Left word / Neutral	Neutral	Neutral / Right word	Right word
Useful<>Useless					
Pleasant<>Unpleasent					
Bad<>Good					
Nice<>Annoying					
Effective<>Superfluous					
Irritating<>Likeable					
Assisting<>Worthless					
Undesirable<>Desirable					
Raising Alertness<> Sleep-inducing					

Ask Graham which condition this was *

- F
- A
- G
- B
- H

« Back Continue »

42% completed

Figure D.8: Page 2 of the questionnaire that was used to assess the acceptance of the subjects towards each interface design

Appendix E

Pilot results

This chapter covers experimental results from the pilot study performed and the changes made for the real experiment due to insights gotten from the pilot study.

E.1 Results

Each trial of each subject is plotted for the pilot to analyse as much as possible. Subjects performed 3 trials instead of the 5 trials during the actual experiment. Furthermore the mean and the medians are plotted of all the subject trials to see how one of these or both could be meaningful. The definition of the markers is shown in Figure E.1.

The figures in this chapter are different than the experimental results chapter (Chapter F) because the plotting script was later changed to show confidence intervals. I choose to show the raw data the way it was analysed after the pilot rather then in the same format as the actual experiment.

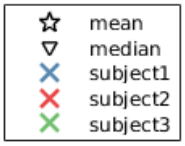


Figure E.1: Legend showing the markers used for each figure in this chapter

E.1.1 Completion time

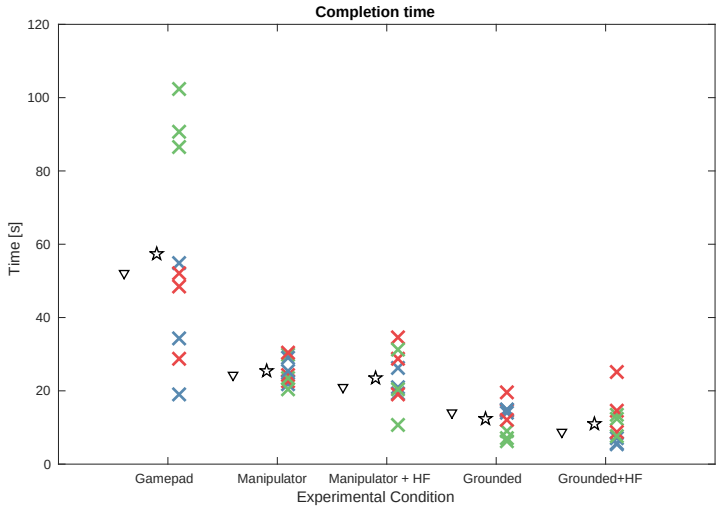


Figure E.2: Pilot results: Completion time metric

E.1.2 Trajectory distance

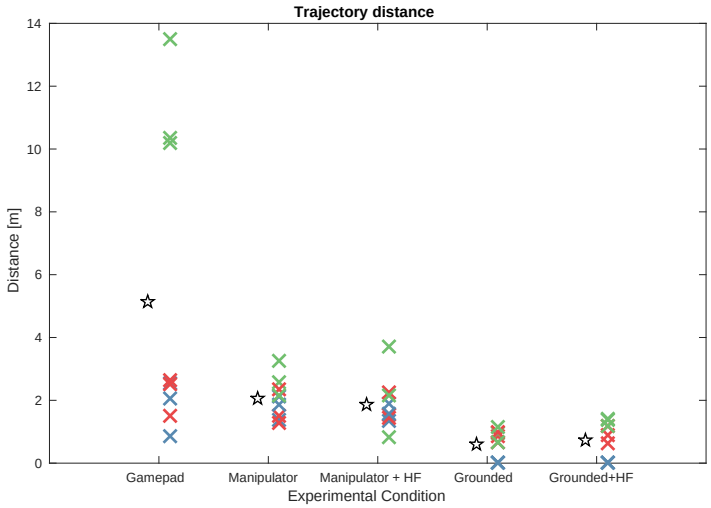


Figure E.3: Pilot results: Trajectory distance metric

E.1.3 Number of flips

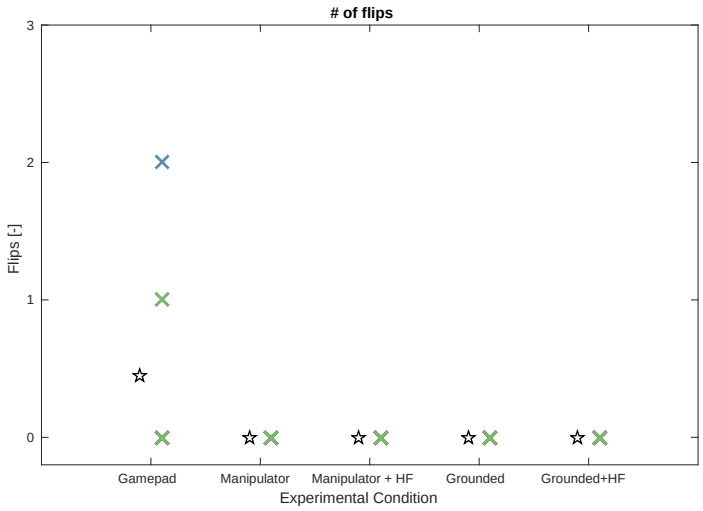


Figure E.4: Pilot results: Cumulative number of flips metric

E.1.4 Standard deviation of the input

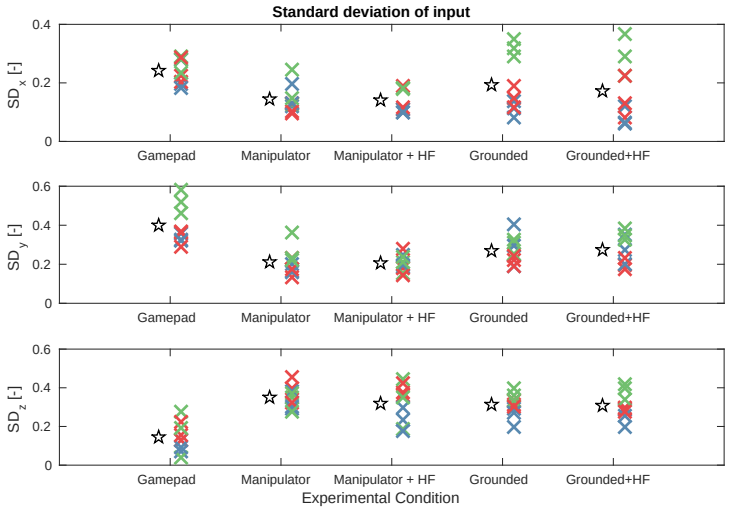


Figure E.5: Pilot results: Standard deviation of the input metric

E.1.5 Number of reversals

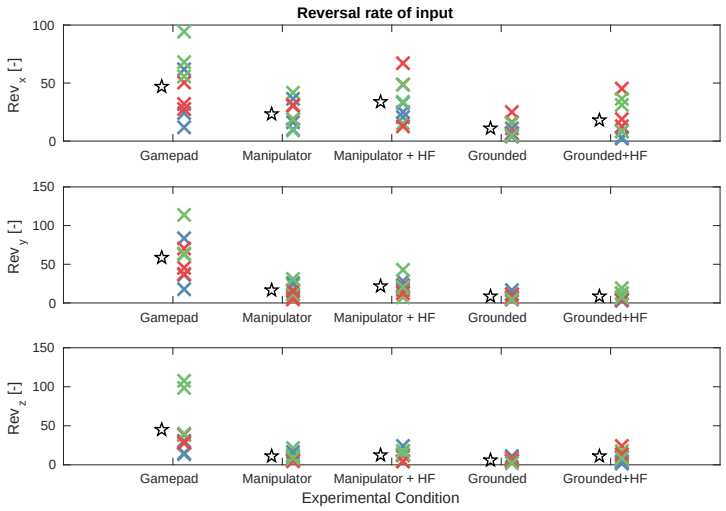


Figure E.6: Pilot results: Number of reversals metric

E.1.6 Average force

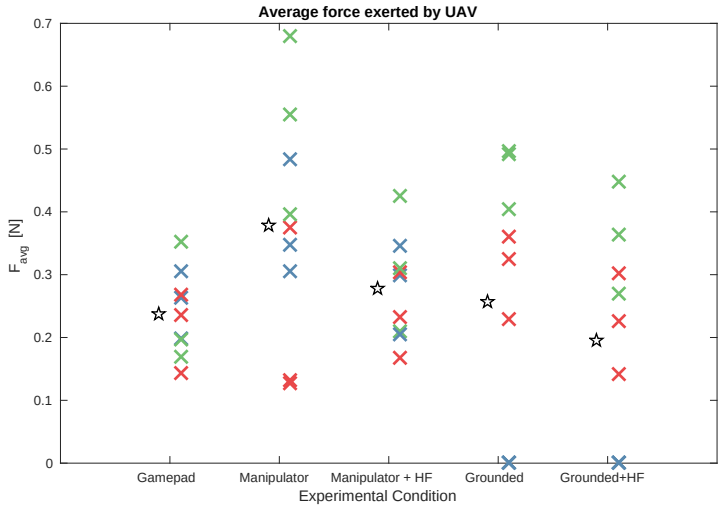


Figure E.7: Pilot results: Average contact force metric

E.1.7 Maximum force

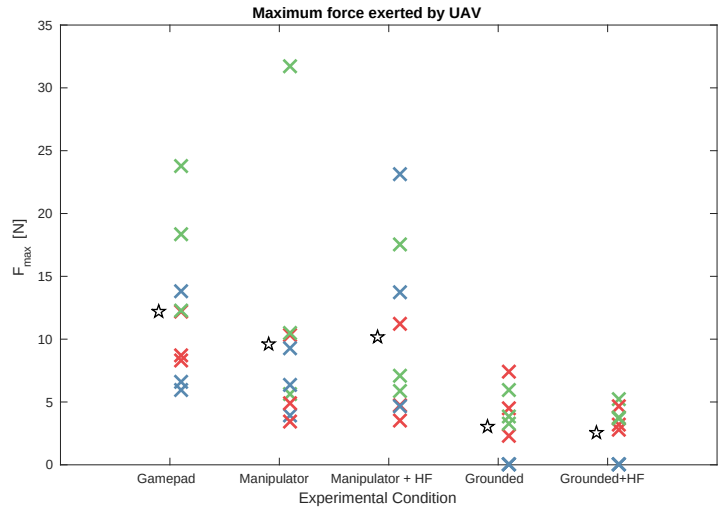


Figure E.8: Pilot results: Maximum contact force metric

E.1.8 Maximum velocity

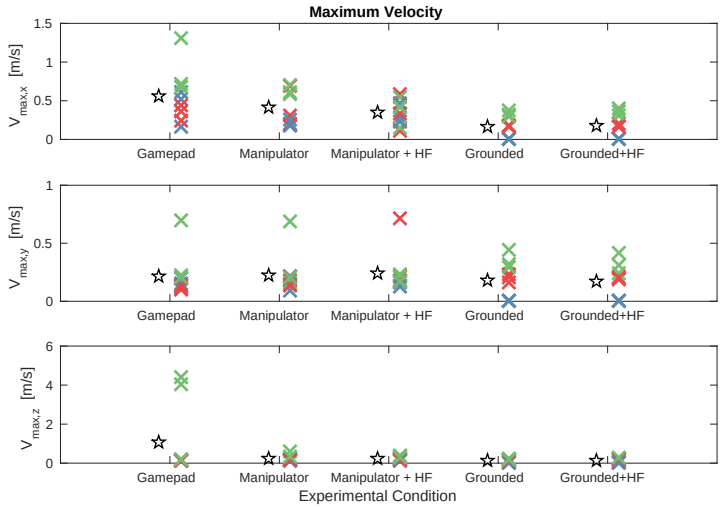


Figure E.9: Pilot results: Maximum velocity metric

E.2 Changes

As is expected of a pilot, some aspects don't go as planned. These aspects were corrected after the pilot study and implemented for the real experiment.

E.2.1 Slave position not saved

Due to a mistake in the `launch` file that started the simulation, the UAV sensors were turned off in the grounded slave conditions which also loaded the UAV position publishing node. Because of this, the slave position was not recorded during the pilot. Additionally, because the location of the UAV was not known in these conditions, the `rosfalcon` node which turned off feedback when the UAV was close to the platform, did not generate any haptic feedback at all.

E.2.2 Cover angle

In addition to the switch angle which is recorded from the topic `/task_status` the cover angle seemed to also be of interest as it was the cover that seemed to be the most difficult part of the task. Therefore the angle of the cover was also recorded under the topic name: `/cover_status`.

E.2.3 Contact transactions

The idea of having a primary and secondary metric for performance was born shortly after the pilot. Completion time reflects very well how effective the task is performed but there is more. It was very clear during the trial that some subjects were able to perform the task relatively well in one go whereas other kept bumping up against the cover and switch without result. The metric for number of contact transitions was thus introduced.

E.2.4 Inefficient time

As is mentioned above in the motivation for adding the metric for the number of contact transitions, I saw a lot of subjects bumping against the cover and the switch without actually effectively moving the switch. The contact point had to be just right and for this reason it was expected that taking a measure for the time that the UAV was in contact but *not* actually performing manipulation, was expected to be a good secondary measure for performance.

E.2.5 Training game

First the subjects were instructed to just "play around" with the interface to get accustomed to the simulation view, the slave dynamics and the master device. This seemed to put subjects in a situation in which they were unsure not only what to do but also when they had "completed" the training. Because of this, a training world was created to keep the subjects engaged (Figure E.10).

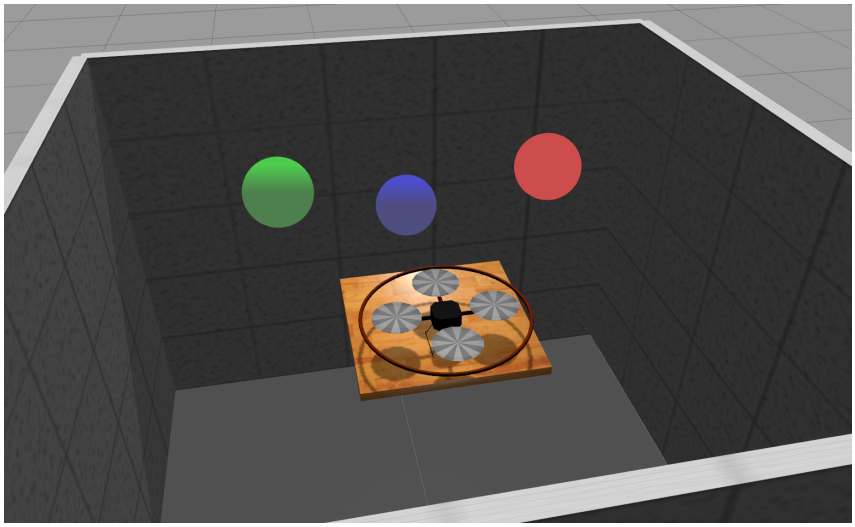


Figure E.10: Training world to engage the subjects during familiarization with the simulation and the master devices. The coloured targets have no collision geometry.

E.2.6 Start further back

It was both noted by me and mentioned by a subject during the pilot that the manipulator was very close to the edge of the workspace at the front (y-axis). The UAV was set to spawn at a constant distance from the switch leaving just as much room at the front of the workspace as at the back. But because there was only a need to fly further forward from the starting point, an asymmetric workspace was implemented where the UAV started 5 cm further back.

E.2.7 Updated training protocol

During he pilot, each subject got precisely 2 training trials per experimental condition. However it became apparent that some were still flipping the UAV on the second trial and thus needed a few more trials. Therefore a variable number of training sessions was allowed, with a maximum of 5.

E.2.8 Simplified instruction video

The instruction video revealed a number of details about the experimental protocol that were not necessary for performing the task and could potentially confuse subjects. The experimental protocol was shown in full, with all its trainings which is a lot of information. Therefore this image was changed to only include the actual experimental conditions. Also because the training level was added instead of just flying around, a video of this was added to the instruction video.

E.2.9 Instructions careful instead of efficient

Initial instructions were to perform the task as fast and as efficient as possible. Though efficiency is important, it does not provide a well balanced intention together with fast. Because we are simulating a task in a disaster area, *caution* is much more important then efficiency. If subjects are not informed to be careful, they will tend to be too focused on speed which would likely lead to more flips. Therefore a better instruction with a realistic trade-off is to be fast and careful instead of fast and efficient.

E.2.10 Limit to one task only

As mentioned in the implementation chapter (section B.1), the initial plan was to test 3 and later only 2 manipulation tasks. The pilot study already took up close to an hour per subject, excluding questionnaire time. In addition, the cover part of the safety switch task is similar to the lever task, a comparable exercise is tested. And finally because the goal of this study was to focus on varying the interface design, and not the task, the former should receive the most focus.

Appendix F

Experimental results

This chapter consists of details results of the human factors experiment. The paper only shows figures of the most important results whereas this chapter shows figures from all the metrics considered as well as raw data from the questionnaire.

F.1 Participants

Subjects were given a codename to make it easy to refer to a certain subject. Only I have their real names to ensure anonymity, as is promised in the informed consent form. The table below shows the personal data of all the subjects which was collected during the questionnaire before and during the experiment (Table F.1).

Name	Gender	Age	Gaming [hrs/wk]	Gaming affinity [-2,2]	UAV Pilot experience
Joel	Male	23	0	1	No
Rein	Male	23	2.5	1	No
Joey	Male	36	0	-1	Yes
Paul	Male	24	0	1	Yes
Klaas	Male	24	0	-2	No
Rachel	Female	24	0	-2	No
Anne	Female	24	0	-2	No
Yalmlar	Male	26	0	0	No
Clark	Male	24	8	1	No
Myrth	Male	25	0	-2	No
Laura	Female	23	3	1	No
Josh	Male	24	0	-1	No
Maxim	Male	26	0	-2	No
Nico	Male	27	0	0	No

Table F.1: Personal data collected of all the subjects that participated in the experiment along with the results of the first part of the questionnaire. Code names are used throughout the experiment for all objective and subjective data to protect their identity. Gaming affinity was assessed using a likert scale ranging from -2 to 2.

F.2 Results

Since more metrics were calculated and graphed than those that were shown in the scientific paper, this chapter gives an overview of all results from all the metrics, in one figure per metric. Just like in the paper, averaged per condition are shown with their 95 % confidence intervals along with individual subject means indicated with crosses. More information on the calculations of these metrics can be found in the metrics chapter (Chapter D).

F.2.1 Completion time

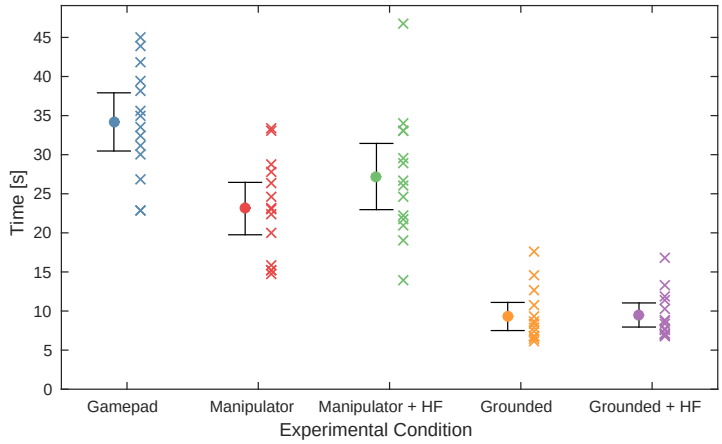


Figure F.1: Experimental results: Completion time metric

F.2.2 Trajectory distance

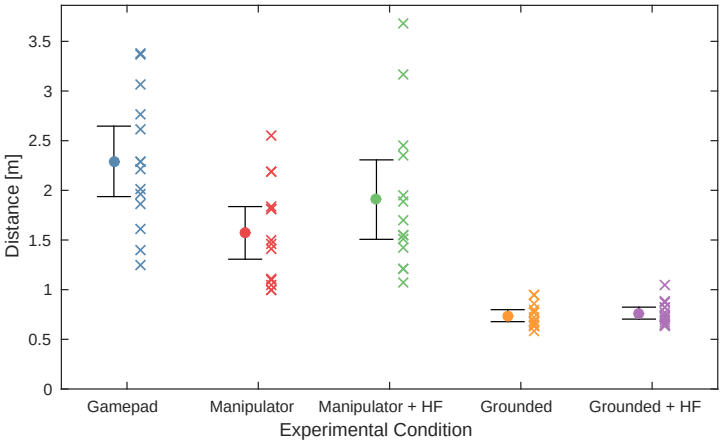


Figure F.2: Experimental results: Trajectory distance metric

F.2.3 Number of flips

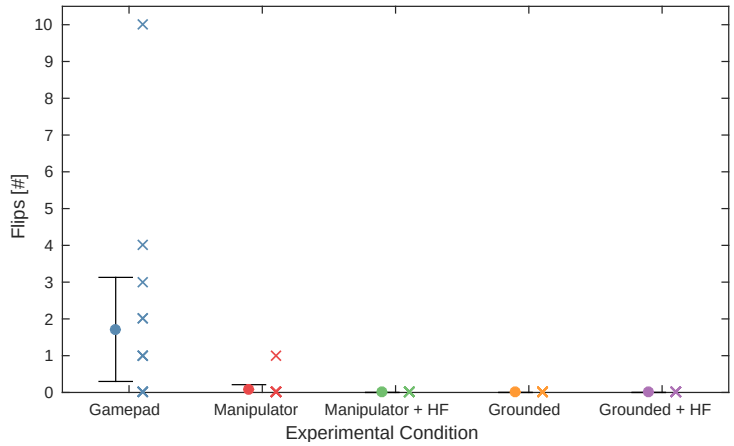


Figure F.3: Experimental results: Cumulative number of flips metric

F.2.4 Standard deviation of the input

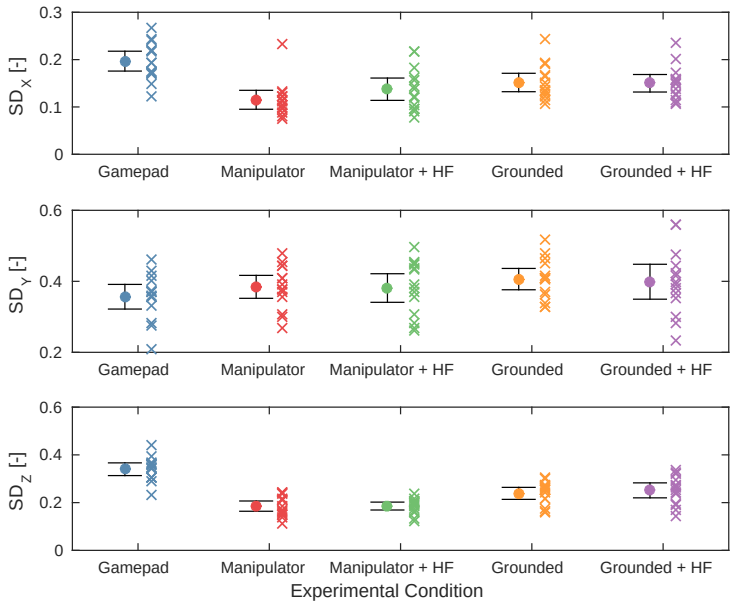


Figure F.4: Experimental results: Standard deviation of the input metric

F.2.5 Number of reversals

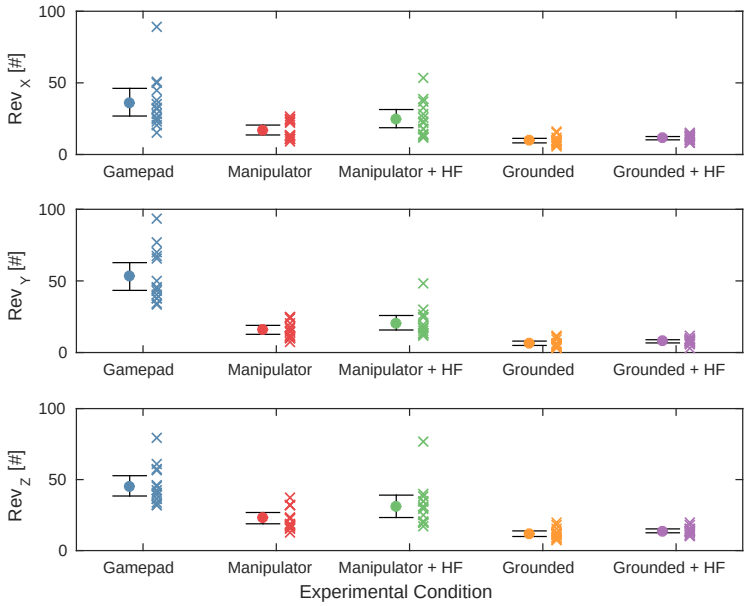


Figure F.5: Experimental results: Number of reversals metric

F.2.6 Average force

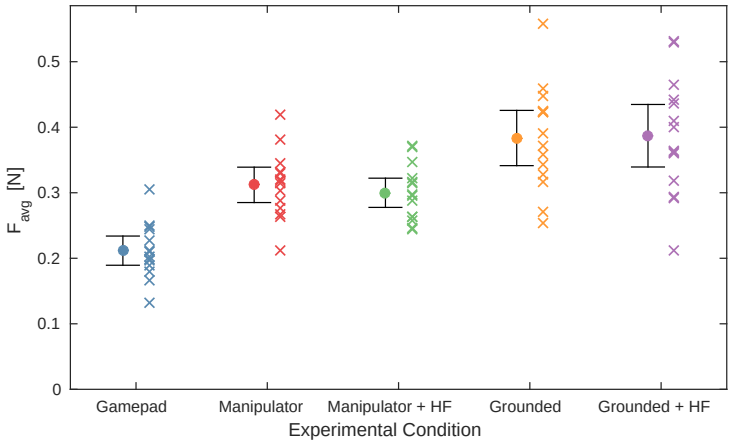


Figure F.6: Experimental results: Average contact force metric

F.2.7 Maximum force

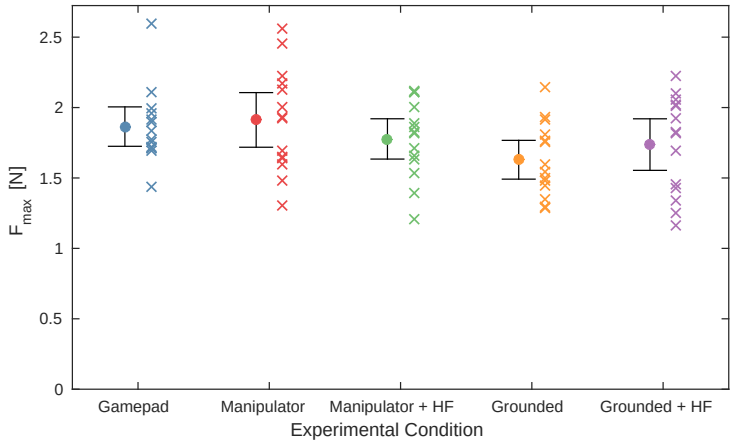


Figure F.7: Experimental results: Maximum contact force metric

F.2.8 Maximum velocity

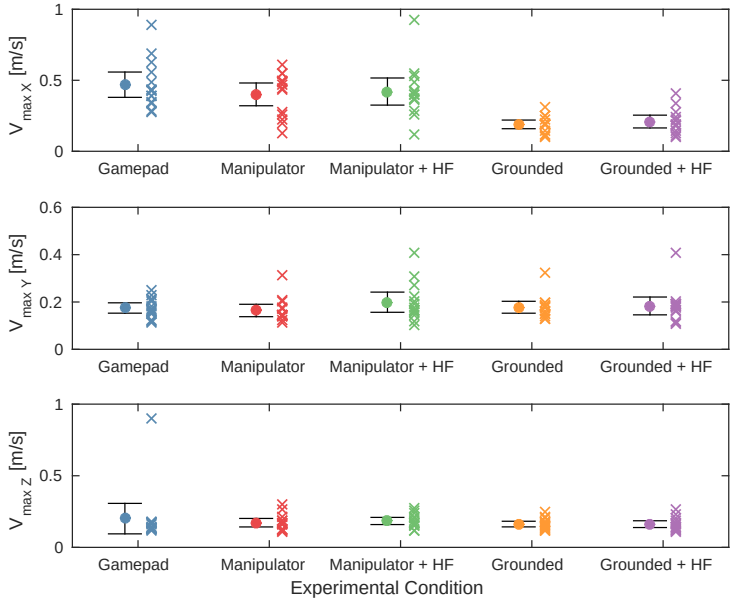


Figure F.8: Experimental results: Maximum velocity metric

F.2.9 Number of contact transitions

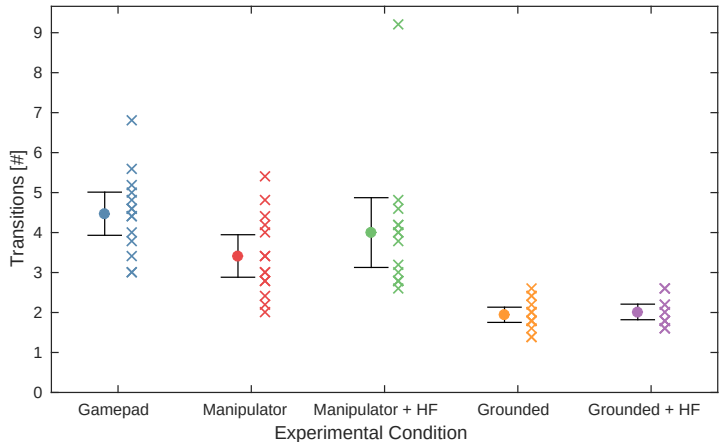


Figure F.9: Experimental results: Number of contact transitions metric

F.2.10 Inefficient time

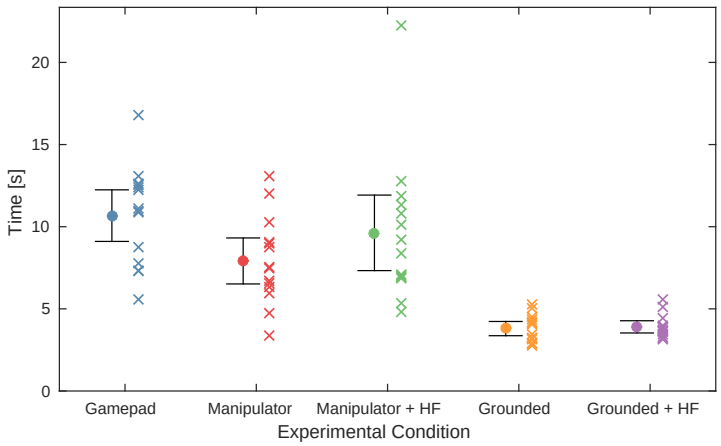


Figure F.10: Experimental results: Inefficient time metric

F.2.11 Subjective acceptance questionnaire

The usefulness and satisfaction scores are computed per subject following the Van Der Laan method [14]. The raw usefulness and satisfaction scores are shown below (Table F.2).

	Gamepad		**Manipulator + HF**		**Manipulator + HF**		**Grounded**		**Grounded + HF**	
Subject	Usefulness	Satisfaction	Usefulness	Satisfaction	Usefulness	Satisfaction	Usefulness	Satisfaction	Usefulness	Satisfaction
Joel	-0.8	-1.75	-0.6	-1	1	1	0.4	0.75	1.4	1.5
Rein	0.4	0.5	0	-0.75	0.4	-0.75	1	1.25	1	1.5
Joey	-0.2	-0.75	-0.2	-0.25	-0.4	-0.75	1.6	2	1.4	1
Paul	1.2	1.25	1.4	2	1.4	2	1.6	2	1.2	1.75
Klaas	-0.6	-0.75	0.2	0.5	1.6	1.25	1.2	1	1	2
Rachel	-0.6	-1.75	1	1	0.8	0.5	1.4	1.25	1.8	1.75
Anne	-1.4	-2	-0.4	-1.5	-0.4	-1	1	1.25	1.4	2
Yalmlar	0.6	0	0	-0.5	1.6	1.25	2	0.5	1.2	1.25
Clark	0.6	0.5	0.6	0.25	0.2	-0.75	0.4	0.75	1.2	0.25
Myrth	0.2	-0.75	0.8	1.25	1.4	1	0.8	1.5	1.6	0.75
Laura	0.2	0.75	1	1.25	0.6	-0.5	0.4	0.5	1	1.75
Josh	-1.4	-2	0.6	1.25	0.8	1.5	0.4	1.25	0.4	0.25
Maxim	-1	-1	0.4	0.25	-0.4	0	0	1.25	1	0
Nico	0.2	-1	1.2	1	1.8	1.25	1.6	0.75	1.8	2

Table F.2: Raw data from subjective acceptance questionnaire of all subjects

Bibliography

[1] J. M. O'Kane, *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, 2013.

[2] I. Baranov, "ROS 101: Intro to the Robot Operating System," 2014. [Online]. Available: http://www.clearpathrobotics.com/2014/01/how-to-guide-ros-101/

[3] "ROS Answers: Open Source Q&A forum." [Online]. Available: http://answers.ros.org/questions/

[4] "Gazebo Answers: Open Source Q&A forum." [Online]. Available: http://answers.gazebosim.org/questions/

[5] "libnifalcon: Open Source driver for the Novint Falcon." [Online]. Available: http://qdot.github.io/libnifalcon/

[6] "GrabCAD - Toggle-switch-guard." [Online]. Available: https://grabcad.com/library/toggle-switch-guard--1

[7] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. Von Stryk, "Comprehensive simulation of quadrotor UAVs using ROS and Gazebo," *Simulation, Modeling and Programming Autonomous Robots (Lecture Notes in Computer Science)*, vol. 7628 LNAI, pp. 400–411, 2012.

[8] A. Seugling and M. Rolin, "Evaluation of physics engines and implementation of a physics module in a 3d-authoring tool," *Umea University*, p. 89, 2006.

[9] T. Erez, Y. Tassa, and E. Todorov, "Simulation Tools for Model-Based Robotics : Comparison of Bullet , Havok , MuJoCo , ODE and PhysX," *IEEE International Conference on Robotics and Automation*, pp. 4397–4404, 2015.

[10] "Gazebo : Tutorial : Model plugins." [Online]. Available: http://gazebosim.org/tutorials/?tut=plugins{_}model

[11] Queensland University of Technology, "RosFalcon - Robotics@QUT - Confluence," 2010. [Online]. Available: https://wiki.qut.edu.au/display/cyphy/RosFalcon

[12] "Robotics System Toolbox - Matlab and Simulink." [Online]. Available: http://nl.mathworks.com/products/robotics/

[13] "bag2mat source file - GitHub." [Online]. Available: https://github.com/osudrl/atrias/blob/master/software/atrias/scripts/bag2mat.py

[14] J. D. Van Der Laan, A. Heino, and D. De Waard, "A simple procedure for the assessment of acceptance of advanced transport telematics," *Transportation Research Part C: Emerging Technologies*, vol. 5, no. 1, pp. 1–10, 1997.