

SEMESTER PROJECT

Title: Adaptive Human Robot Interface for Aerial Robots

Student(s): Tristan Bonato (Robotics)

Professor: Dario Floreano

Assistant 1: Matteo Macchini

Assistant 2: Davide Zambrano

Project description:

In the Laboratory of Intelligent Systems, we are working on the development of motion-based intuitive interfaces for the control of mobile robots. Such interfaces are based on the observation of people's instinctive body motion while observing robot movements. Despite the high intuitiveness of motion-based interfaces, users would profit of an adaptive systems able to cope with their changes of preferences over time. For this reason, we are studying the design of an adaptive interface which learns online the user's behavior during teleoperation and changes accordingly, to simplify the task. For this project, you will implement a system that learns autonomously how to fly a fixed-wing drone in simulation, then adapt its behavior to match with real people executing the same task. Finally, you will test the system with an adaptive interface and evaluate its effectiveness both in simulation and with real operators.

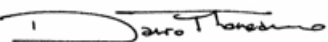
Students interested in aerial robotics, human-robot interfaces and machine learning are encouraged to apply.

Remarks:


You should present a research plan (Gantt chart) to your first assistant before the end of the second week of the project. An intermediate presentation of your project, containing 8 minutes of presentation and 7 minutes of discussion, will be held on October 27, 2020. The goal of this presentation is to briefly summarize the work done so far and discuss a precise plan for the remaining of the project. Your final report should start by the original project description (this page) followed by a one page summary of your work. This summary (single sided A4), should contain the date, laboratory name, project title and type (semester project or master project) followed by the description of the project and 1 or 2 representative figures. In the report, importance will be given to the description of the experiments and to the obtained results. A preliminary version of your report should be given to your first assistant at the latest 10 days before the final hand-in deadline. 2 copies of your final version, signed and dated, should be brought to your first assistant before noon January 15, 2021. A 20 minute project defense, including 5 minutes for discussion, will take place January 27, 2021. You will be graded based on your results, report, final defense, and working style. All documents, including the report (source and pdf), summary page and presentations along with the source of your programs should be handed-in as a single compressed file on the day of the final defense at the latest.

Responsible professor:

Responsible assistant:

Signature: 

Dario Floreano

Signature: 

Matteo Macchini

Lausanne, 11 September 2020

Adaptive Human Robot Interface for Aerial Robots

Tristan Bonato
School of Engineering (STI)
École polytechnique fédérale de Lausanne (EPFL)
Lausanne, Switzerland
Email: tristan.bonato@epfl.ch

Abstract—In this project, we study the human-robot interface for aerial robots. The goal of the project is to create an adaptive interface that will tailor the gains of a drone, according to human behaviour, while piloting. In our case, the gains are the sensibilities of the controller for the roll and the pitch, but it can be extended to a different application in teleoperation with various mobile robots.

This project is a continuation of three other projects. In these projects, they designed a fixed-wing drone simulator on unity (3D Game simulator). In this simulator, they did an algorithm that implements spline between waypoints and PID control. With this structure, they worked on a deep reinforcement learning to match human behaviour and PID autopilot.

They also recorded human data teleoperation online with trackpad and mouse control.

Our study uses their previous structure and learnings. We understand that to match human and autopilot behaviours, we need to have similar data. Thus, we focused on this study to create a mathematical spline as similar as possible to human trajectory and adapted the dynamics of the autopilot to human behaviour. With that, we implemented a Linear regression to compare both input data and improve human teleoperation.

The results of this study give information on human behaviour in teleoperation. It also shows how the interface drive-by iterative linear regression works.

I. INTRODUCTION

A. Context

Aerials robot's industry increases significantly every year. It touches a lot of different applications, including rescue mission and exploration in outdoor or indoor environment. For most of them, we need teleoperation to control them. It implies some problems like communication delay or accuracy issues. Furthermore, how a human drive a drone depends strongly on the experience of the user in teleoperation, the knowledge of the robot to control and the time spend to practice.

In this project, we want to decrease the time needed to drive a mobile robot efficiently and increase the accuracy of the users, depending on the personal ability, experience and skills of the users. To reach this goal, we create an interface that will adapt the sensibilities of the controller to the user, in regard to his personal skills, during the flight. With this interface, the user will learn how to drive the drone, and at the same time, the system will adapt itself to the user.

B. State of the Art

Joystick is not the only way to control mobile robots. A study from Jenifer Miehlebradt and al.[Miehlebradt] has developed a body motion interface to control the drone. Another

study, with body motion, is made by Matteo Macchini and al [Matteo]. They used the body-machine mapping to provides a personalized body-machine interface.

The results of these studies show that with these methods, the learning time decreases and the accuracy in teleoperation is improved.

Another interesting study is the myoelectric based human-machine interface [Oskoei]. They applied a myoelectric based human-machine interface (HMI) to a video game. The signals emitted by the muscle will control the video game. The goal of this study is to create an interface that keeps good performance in long-term operation, with all kinds of users and even if the user is tired. The interface adapts itself in function of the time, the tiredness and the user. They applied an iterative algorithm to learn in real-time the evolution of the electric signal.

The previous work made by the students of the LIS lab [Zhou], [Gregoire],[Baudoin]. They implemented an adaptive human-robot interface between the controller and the robot that will, in real-time, modify the user command and send it to the robot. The learning phase is made offline by reinforcement learning.

C. Objectives

This project has the purpose of giving a new way to create an adaptive interface. The first work failed because the correlation between human inputs and autopilots inputs was not correlated. The main goal that differentiates from the past studies is to do all the computation and work online, without a learning process, and a supervised method. To do that, we found a mathematical spline closest as possible from the human trajectory. We worked on the dynamics of autopilots to have inputs from autopilots similar to human inputs. With that, we have an "optimum" trajectory and "optimum" inputs from autopilots that follows the path. Thus, with an iterative linear regression, we compare at some point in the path in real-time human input to optimum inputs from autopilot and adapt the gains in function, to help the user to improves his drive. The idea is to adapt the gains to help human to have a flight similar to the autopilots.

II. METHODS

To create the adaptive interface, we separate the work in three main steps. The first one is to create a spline between waypoints that look the most as possible to human trajectories. After, we applied an autopilot that follows the spline with a dynamic closest as possible to humans. These two steps give us

a path close to what human does and the inputs from autopilots to achieve these paths perfectly.

And finally, we implemented an iterative linear regression that will adapt the gains to the user online. The purpose is to correct the human inputs and trajectory to look like the autopilots do. All these works are done with two different tools. We use unity for the drone environment and simulation. Unity is a 3D game development in C sharp. It is used to implement the spline, the PID controller, set all the parameters and has other utilities. Then we use a python Notebook to receive and send information from unity.

A. Spline calculation

The goal of this section is to create a mathematical spline, that resembles the human trajectory. In our project, we have 42 waypoints. We need to have a spline that passes through all of them and have enough parameters to tune the curvature between each point. The last assumption is needed to adapt the curve to a human path. We used Catmull-Rom spline for that. This algorithm created a smooth polynomials function (1) between waypoints and have a parameter tau to modify the tension of the curve.

$$\mathbf{p}(s) = \mathbf{c}_0 + \mathbf{c}_1 u + \mathbf{c}_2 u^2 + \mathbf{c}_3 u^3 \quad (1)$$

$$\mathbf{c}_0 = \mathbf{p}_{i-1}$$

$$\mathbf{c}_1 = \tau (\mathbf{p}_i - \mathbf{p}_{i-2})$$

$$\mathbf{c}_2 = 3 (\mathbf{p}_i - \mathbf{p}_{i-1}) - \tau (\mathbf{p}_{i+1} - \mathbf{p}_{i-1}) - 2\tau (\mathbf{p}_i - \mathbf{p}_{i-2})$$

$$\mathbf{c}_3 = -2 (\mathbf{p}_i - \mathbf{p}_{i-1}) + \tau (\mathbf{p}_{i+1} - \mathbf{p}_{i-1}) + \tau (\mathbf{p}_i - \mathbf{p}_{i-2}) \quad (2)$$

The mathematical formula 1 needs 4 points and Tau parameters. However, it creates a path just between the two middle points. The first point and the fourth point are used to adapt the curve between the two middle points. To make a path with a lot of points, we used the first 4 points (P1, P2, P3 and P4) to create a path between P2 and P3. Then, we shift all the point to the next one (p1 = p2, p2 = p3 etc.) and do the operation again and so on.

The equations (2) used the parameter Tau to modify the tension of the curve. If Tau is close to 0, the tension is strong, and if Tau is close to 1, it is the inverse. You can see its effect on the Fig.1.

Finally, we have also the resolution between two points that can change the number of points between two waypoints. More resolution you have, more step between two waypoints will be sent to the autopilots before reach a waypoint.

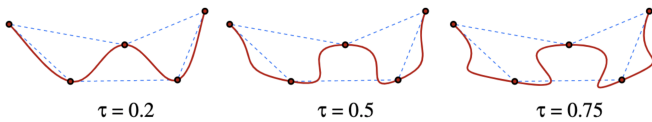


Fig. 1. Effect of the tension parameter Tau on the curve. The figure on the left has a small Tau. Thus, the tension is high and the lines between points are almost direct. Backwards, the figure on the right has a high Tau. Thus, the tension is low and the lines between two points are curvier.

The previous work gives us seven subject's data. We make a mean of the subjects, and we will compare it to our spline algorithm (with different parameters). We use a mean squared error algorithm for the comparison. To be able to compare efficiently two different parts, we first need to clean the data from the subjects and interpolate the data. Finally, with the same time vector in the functions, we can compare the data efficiently.

The previous study already implemented Catmull-Rom spline, but we modified the code to add the parameter Tau. We supposed that with these parameters, we would adapt better the curve.

We also suppose that creating only one path to follow at the beginning of the simulation and not at every step, will create a smoother curve.

B. Controller Design

This section has the purpose of finding a PID autopilot that has similar inputs to humans. We would point out again that the input for this study is the roll and the pitch. To do that, we first study the behaviour of the subjects. We check the correlation between them, the maximal values and the shape of the inputs.

Then, we want a PID that follows the path efficiently. Thus, we do a grid search in the python notebook. The python will send some PID parameters on the unity. Unity will follow the path and send data to python. Finally, python will record the data and compare them at the end of all simulations. We try to find three different autopilots for P, PI and PD to have different alternatives for the linear regression.

Now with all this data, we compare them to humans to see if the inputs from our PID look like the mean of all human inputs.

C. Linear Regression

In this section, we used the previous work to implement the linear regression. The drone is drove by a user. But at each step, the autopilot algorithm is used to know which inputs it would use if it has the control of the drone. This data is not used to control the drone but to be compared with human's inputs with the purpose to modify the interface.

This section is split into two parts. The first part will explain how the system works. The second part explains each development step to arrive at the final result.

1) *The system:* The Human-machine interface architecture is made like in figure 2. The Human will drive the drone by sending the desired roll and pitch outputs from the remote controller. The interface, a matrix 2x2 with the gain roll and the gain pitch, will multiply the displacement of the remote controller before sending it to the drone (Eq.3). The interface is updated each time the drone passes through a waypoint. Thus, the gains are modified at this moment.

$$\begin{bmatrix} y_{t,roll} \\ y_{t,pitch} \end{bmatrix} = G_t \cdot \begin{bmatrix} x_{t,roll} \\ x_{t,pitch} \end{bmatrix} \quad (3)$$

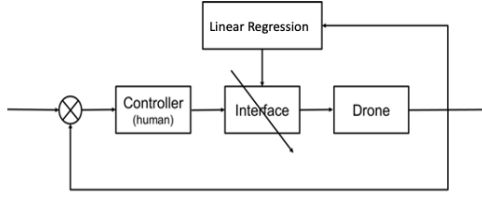


Fig. 2. Human machine interface architecture. The interface is placed between the controller and the drone. At each step, the interface is updated.

The interface uses two different programs. Unity receives information of the user, performs the autopilot and simulates the drone and environment. The linear regression is implemented on python.

The process of communication between unity and python is represented in figure 4. Firstly, an initial linear regression is made with the initial gains chosen on unity and some hypothetical points. Then at each step, unity sends the information of the remote controller, the input from the autopilot and a state. The state informs the drone whether it has reached a waypoint. With this information, the iterative linear regression is updated (if we are through a waypoint) and the output of pitch and roll is computing and send to the drone.

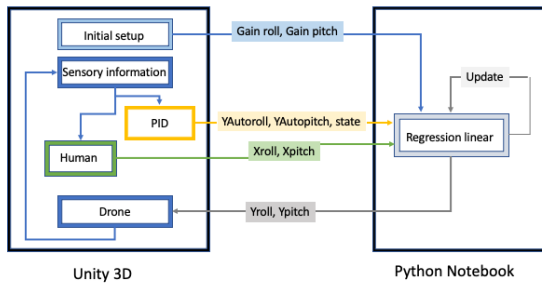


Fig. 3. Process of communication between unity and python. Unity sends information of the remote controller, of the autopilot and of the drone passe through a waypoint. The python notebook with the help of the linear regression, calculate the roll and pitch and send it to unity to control the drones.

To update the linear regression, we take the displacement of the remote controller and coupled it with the autopilot hypothetical values. This step will modify the gain.

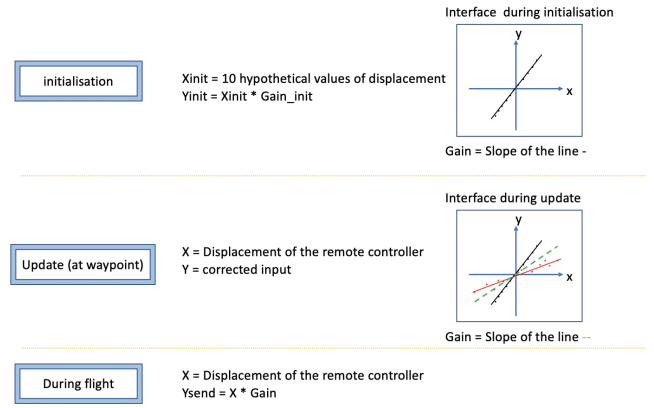


Fig. 4. This figure shown a schema of the three steps for the linear regression. The first one is the initialization. It is created with 10 points between [-0.5 and 0.5] and the initial gains in unity. The second step is the updated. Each time the drone pass through a waypoint a new point is add to the linear regression that will modify the gain. The last one happens every time step in the simulation. It takes the displacement of the remote controller and multiplied it by the gain to have the final input (roll or pitch).

2) *Process*: This section explains the methods we used to reach the implementation of the iterative linear regression. The schema on the figure 5 illustrates the process.

Firstly, we run the simulation with a mouse for the remote controller, and we follow the path the best we can. During the flight, we collect the hypothetical input from the two different autopilots (PD and PI). We look at the data and check if there is a correlation between the human inputs and the autopilots inputs.

We also use another strategy. The human acts like a kind of controller itself. It can be wrong to add a complex controller to another. Thus, we use the inputs from the human and add an error (multiplied by a gain) (Eq. 4). The error used is the same one used for the PID controller, An error quaternion. It takes in count the orientation of the drone and the desired orientation and calculate the variation needed to correct the trajectory. Thus, the error, multiplied by a gain behaves, is simply a P controller. But it is important to understand that this P controller has not the purpose to follows the path itself. It has to purpose to be added to the human input, to correct the human input in order to look like a PID that follows the path perfectly.

To tune the P gain a grid search method is used. We run the simulation a few times with the inputs from an advanced user and different gain P. At the end of all the simulation, we compare the corrected user inputs with the inputs from autopilot found in section II-B.

$$\text{Inputs}_{\text{optimal}} \approx \text{Inputs}_{\text{Human}} + \text{Error} * P_{\text{Gain}} \quad (4)$$

After that, we finalize the tuning of the data. It means checking if the data are in phase or if we need to add a filter. Before doing the experiment, we tried the interface with a member of the study. We made some flights with different

gains and checked if the interface converges to the same gain after some runs. Finally, to confirm the results, we would have had to test the interface with different subjects and analyze the data.

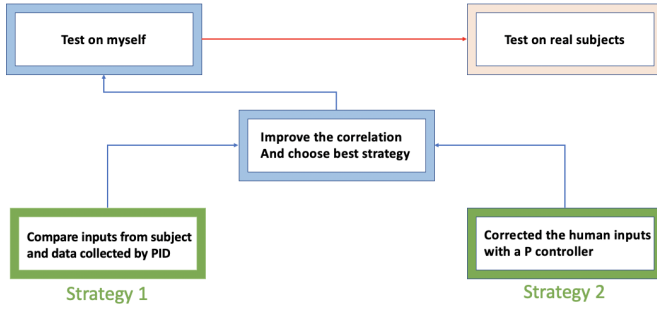


Fig. 5. Diagram of the process to reach the objective. First, in the green rectangle, we have two strategies. First, one collects PI and PD inputs during a human flight and compare inputs from autopilot and human. The second one collects P inputs during a human flight. Then, it corrects the human input with the help of a P controller. And finally, we compare the corrected flight with the inputs of a PID controller that follows perfectly the path. After this step, we improve the correlation if necessary, for strategy one and two. And we choose the one with better results. Thus, we run the simulation with the interface and a human user and check if the linear regression has relevant data. Finally, to confirm the previous results, we have tested the interface with a user from the lab and finally with different subjects.

3) *Experiment*: We want to know how linear regression is working. We had to verify if the algorithm converges to an adapt gain depending on the subject. We also want to know if the subject learns faster with the interface and if it's more pleasant to drive with the interface or not.

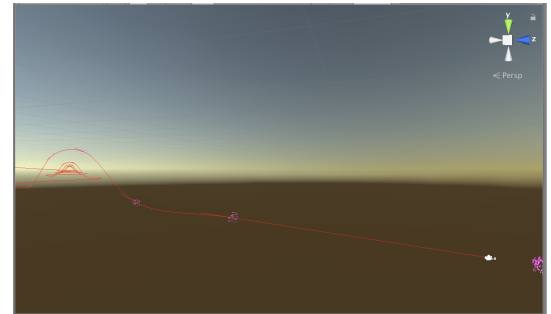
Thus, we do an experiment to test the results. The experiment follows three phases; the baseline to let the user have a hand on with the simulator (2 runs), the training (3 runs) and the evaluation (2 runs). A run is to pilot the drone between 37 waypoints. After each phase, we asked the user what his feelings about the sensibility of the teleoperation are. We used 18 subjects distributed in three groups. The first group does not use the interface; it's the comparison group. The second group has the interface for the training phase. And the third group has the interface for the baseline and the training. The groups two and three are here to understand if the interface provides assistance. And also, to understand after how many steps the interface converges to his optimal gains. Into each group, we have three subgroups with different initial gains; 0.5, 1 and 4. We used these gains because we want to check if the interface can increase or decrease the gains. A gain equal to 0.5 is the lowest gain possible to be able to reach the waypoints. And a gain bigger than four will make the simulation too hard for the user.

III. RESULTS

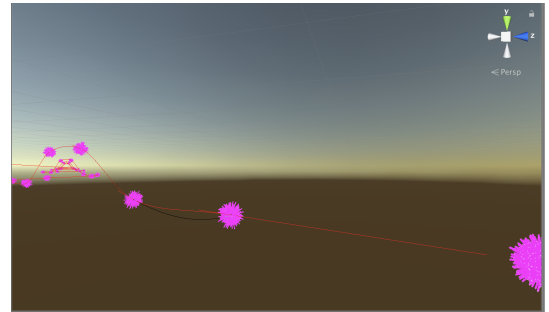
We split the results into the three parts described in the methods plus one part for the experiment with subjects.

A. Comparison between human trajectory and mathematical spline

So far, we modified the previous code to make one path for all the waypoints at the beginning of the simulation and compared the Catmull-Rom spline algorithm with different parameters. The tension parameter τ is changed between 0-1 and the resolution between 5-25. We choose this range for the resolution because 5 is a small amount of point between waypoints and 25 is a big amount of point. We supposed that we could understand the effect of resolution with these values. The previous project updated the spline while piloting. This creates some discontinuity to the trajectory of the drone because at every step another direction (because the path has changed) is sent to the autopilots. This phenomenon is strongly increased when the drone is closed to a waypoint. This is not good for the similarity between human and autopilot trajectory because it will produce a disorder in the input. The Fig.6 show us this phenomenon. We have two lines, the red one is the path for all the waypoint, and the black one is a path that simulates each step between 4 points. We can notice that in the first picture both lines are confounding, but in the second picture we have a strong discontinuity for the black line. In this project, we only compute the path one time at the beginning of the simulation. And then, the autopilot will follow this path during all the flight.



(a) Photo of the path from unity at initialization.



(b) Photo of the path from unity after a waypoint

Fig. 6. Photo of the path from unity. The red line is the path for all the way-point, and the black one is recompute at every step between 4 points.

Now we can focus on the parameters of the Catmull-Rom spline. We make a mean square error between the mean of the subject trajectory and nine different algorithmic splines. The parameters for the spline are a combination of resolution =

[5,15,25] and $\text{Tau} = [0,0.5,1]$. we notice that the best results for this experience are with a $\text{tau} = 0.5$ and enough resolution (more than 15 points) (Fig.7).

We can underline that the factors, unfortunately, do not change the results significantly. Thus, we do not run more experiment for these parameters. Furthermore, it is interesting that $\text{tau} = 0.5$ represents a centripetal Catmull-Rom spline. It is the mathematical spline with the smallest slope and thus the smoothest path.

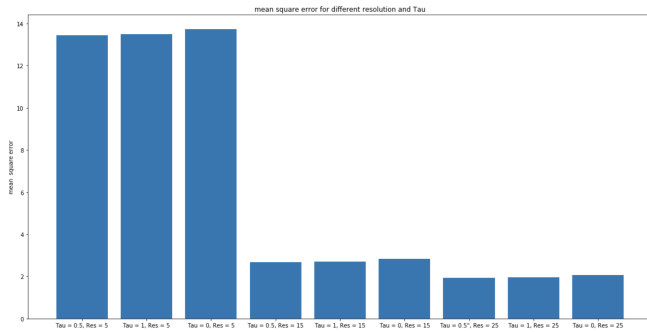


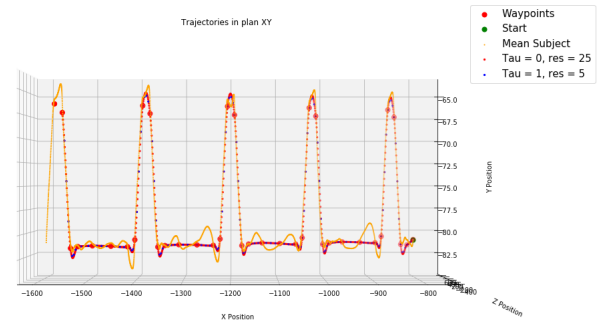
Fig. 7. Mean square Error between the mean of the trajectory from all subject and different parameters for the algorithmic spline. We can remark that the difference is hardly noticeable.

We plotted the path in two different 2D planes to confirm these results, with the smallest error, the most significant error and the mean trajectory (Fig.8). As expected, the algorithmic splines are very similar, and it is challenging to choose the best one with the naked eye. We notice in the same plot that the human trajectory and one of the algorithmic splines are not superposed perfectly. However, close to the waypoints, the paths are similar.

To conclude this part, we choose a Tau equal to 0.5 and a resolution equal to 15 points for the rest of the study. We do not take more significant values in terms of resolution because it will significantly improve the similitude between both paths. Furthermore, different resolution implies more calculation for the computer and less time for the PID to correct its path.

B. Comparison between human input and autopilot input

We begin to analyze the inputs of all the subjects. The plots on the ten first waypoints are on the Fig.9. The plots showed only ten waypoints because the full path is periodic, and to we have better readability. We noticed that all the subjects have almost the same behaviour. The mean of the inputs from all of the subjects can be considered as a new human trajectory. All the inputs are ranging between $[-0.6,0.6]$; the human did not make an abrupt change.



(a) Plot of the paths in the plane XY.



(b) Plot of the paths in the plane XZ.

Fig. 8. Plot of the paths in two different planes. Three paths are represented; the orange one is the mean of the subject, the red one is with tau equal to 0 and resolution equal to 5, and the blue one is with tau equal to 1 and resolution equal to 25. Despite the strongly different parameters, the curve looks the same.

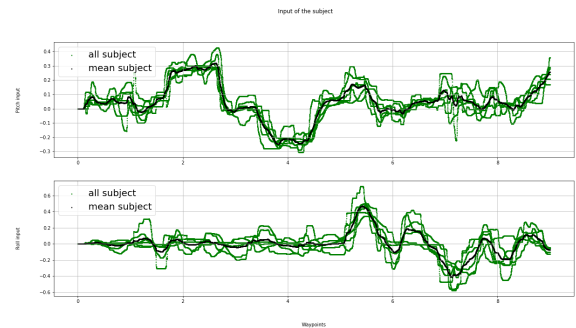


Fig. 9. Plot of the input roll (bottom) and pitch (top) on 10 waypoints, with all the subject in green and the mean of the subject in black

A grid search method optimizes the PID controller. We found a PI and a PD controller to follows precisely the path. It seems impossible to follow the path only with a P controller due to the various curvatures in the track. A P gain too big will creates strong oscillation and instability. Moreover, a P gain too small will not follow the path when the curvature is strong. In the Fig.10, three different curves are shown; the input of PI, PD and the mean of subjects. All of the three curves are very similar and smooth.

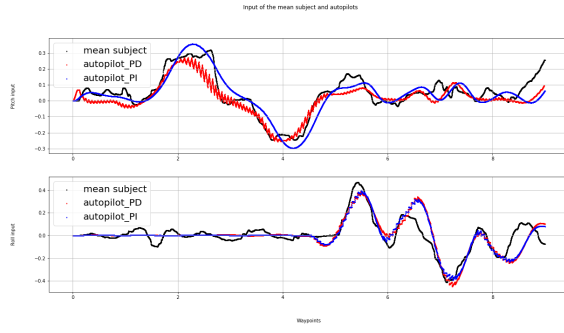


Fig. 10. Plot of the input roll (bottom) and pitch (top) on 10 way-points, with all the mean of the subject in black, the PD autopilot in red and the PI autopilot in blue.

C. Effect of the linear regression

We present the results in three different parts described in section II-C2.

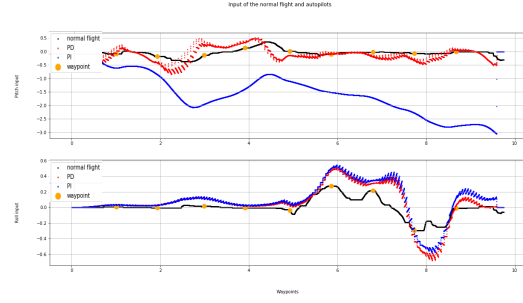
We ran unity with three flights: a flight close to the path, a flight with too big sensibilities and a flight with too small sensibilities. We call them on the plots respectively, normal flight, speed flight and slow flight. These flights simulate three different kinds of users.

1) *Check data from subject and autopilot for the first strategy:* First, let us study the first strategy. The PI and PD autopilots' data are recorded and shown in figure 11.

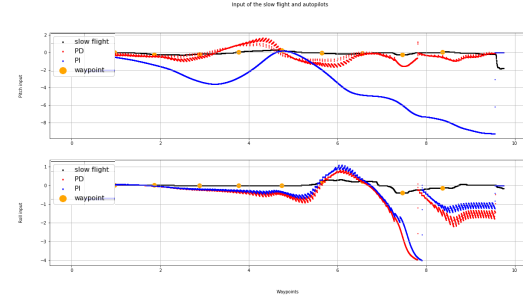
The results on these plots are not sufficient to run the linear regression. We cannot extract a correlation between autopilots and all these flights. A different behaviour implies drastic change. The integrative term is summed without the capacity to change the trajectory. That creates a significant offset. The derivative term adds much noise. It is not possible to use Integrative term and derivative term for our interface.

We tried different ways to improve the correlation without results. For example, we implemented an exponential filter to smooth and decrease the dynamics of the curve. We also thought that making the autopilot look further in the path could help. We ran many different kinds of grid search to find P, PI, PD, or PID autopilots. All these attempts produce the same kind of inputs than presented in fig. 11.

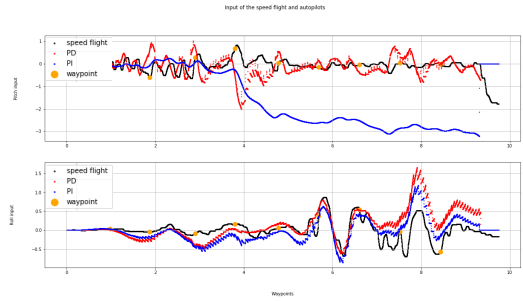
2) *Check data from subject and autopilot for the second strategy:* Thus, we changed our strategy. We added the error to the user's inputs and tuned it again for it to look similar to PID autopilot from section II-B. As a reminder, this PID autopilot can follow the path smoothly and perfectly. For the end of the report, we call it optimal autopilot. The plots on figure 12 show us better results than the previous strategy. We call the error multiply by a gain "autopilot P" in the plot. We can see a correlation between the normal flight and the slow flight. It is difficult for a chaotic flight like the speed flight to compare them because of the inputs' substantial variation. Despite this, this method will be used for the next step of the linear regression.



(a) Plot of the inputs from Pi and PD autopilots and a normal flight.



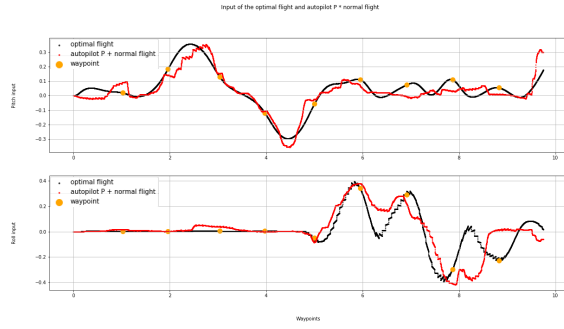
(b) Plot of the inputs from Pi and PD autopilots and a slow flight.



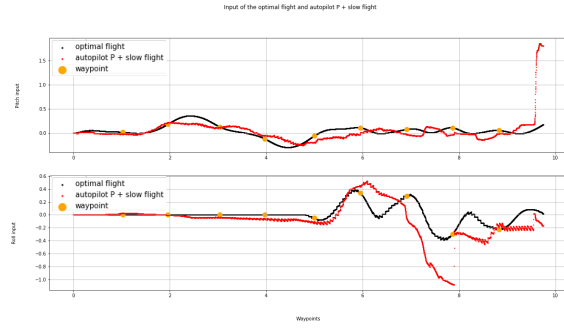
(c) Plot of the inputs from Pi and PD autopilots and a speed flight.

Fig. 11. Plot of the inputs from Pi and PD autopilots for three different flights. In each subplot, you have the pitch on the top and the roll on the bottom. No evident correlation between autopilots and flight can be put on light.

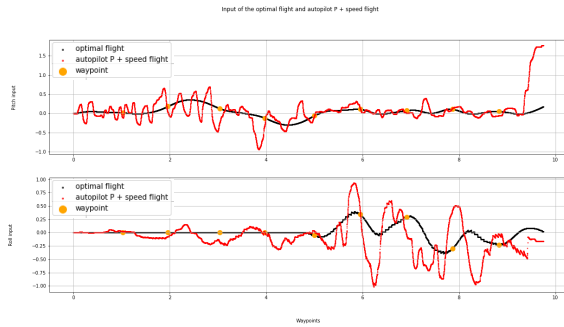
3) *Test on linear regression:* One person has tested the drone with the full interface active. The figure 13 shows that the gains adapt automatically to the user. The plot shows the evolution of the gain after one run. The black dot is points used to initialize the initial gain from unity. The dot of colours is the values made with the remote controller's displacement for the axis x and the inputs of corrected input for the axis y. Two different initial sensibilities are used; 3 and 0.5. These sensibilities are chosen because bigger than 3 it is challenging to drive, and smaller than 0.5, it is impossible to follow the path. The first run is with high sensibility, and the second run is another one initiate with low sensibility. At the end of both



(a) Plot of the inputs from optimal autopilot and a normal flight + error.



(b) Plot of the inputs from optimal autopilot and a slow flight error.

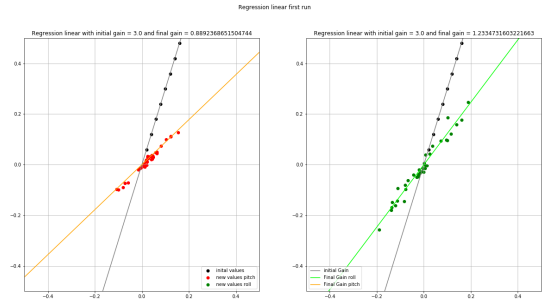


(c) Plot of the inputs from optimal autopilot and a speed flight + error.

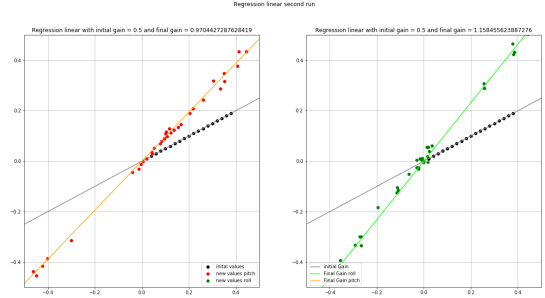
Fig. 12. Plot of the inputs from optimal autopilot for three different flights + error. In each subplot, you have the pitch on the top and the roll on the bottom. A correlation between autopilots and flight can be put on light.

experiences, the sensibilities converge to 0.9 for pitch and 1.2 for the roll. This result shows that the algorithm makes the gains converge to the user's preferred gains.

We make another test with a different speed for the drone. We increase the speed by 50 per cent and see if the algorithm still works. We make the same kind of experience but with two different initial gain; 1 and 3. we choose these values for the same reason that the previous experiment. The algorithm converges with two different initials gain to the same gains; about 1.1 for the pitch and 2.2 for the roll. You can see the



(a) Plot of linear regression with a high initial gains.



(b) Plot of linear regression with a low initial gains.

Fig. 13. Plots of the linear regression with two different initial gains and regular speed. For each subplot, we have the pitch on the left and the roll on the right. The black dots represent initial point for the initial linear regression. Red dots are new values at waypoints. The black line is the initial gain (equal to initial gain) and the red line is the new gain after one runs. After one run, the gain adapts itself to 0.9 for the pitch and 1.2 for the roll. This seems my best gain for driving the drone.

linear regression plot on the fig.14

The results show us that the interface adapts itself to the simulator's configuration and converges to a gain adapts for the user. To confirm the results, we need to test this experience on different subjects.

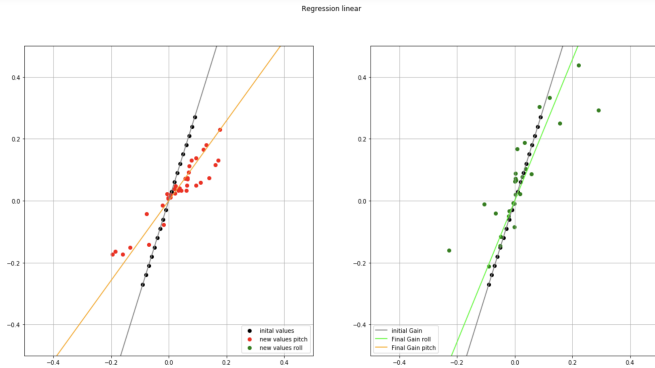
D. Experiment with subjects

The results are separate into three parts. The first one is to understand if the interface helps the user to learn faster. The second part shows if the user has more pleasure to drive with the interface or not. Moreover, the last parts show some characteristic of the interface.

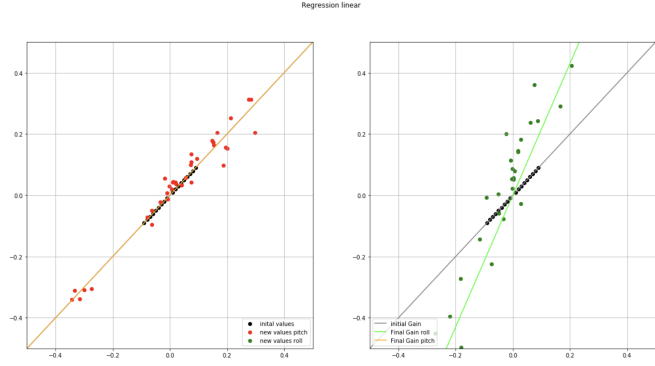
1) *Comparison of performance*: To compare the different groups' performance, we make a mean of the error for each group at each phase. Then we calculate the proportionality between the mean of the baseline and the mean of the evaluation. In figure 15, we can see a plot of the results.

2) *Comparison of satisfaction*: After each phase, we asked the user if it feels a difference with the gain. The survey shows that people feel an improvement in gain with or without the interface.

3) *characteristic of the interface*: The interface has interesting results. After the experiment, we can confirm that the interface converges to a final gain for the pitch and the roll.



(a) Plot of linear regression with a high initial gains.

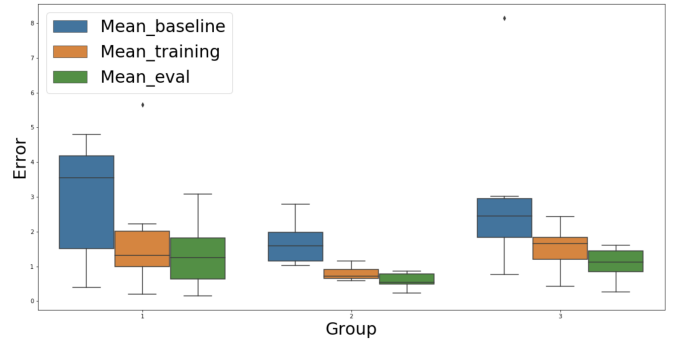


(b) Plot of linear regression with a low initial gains.

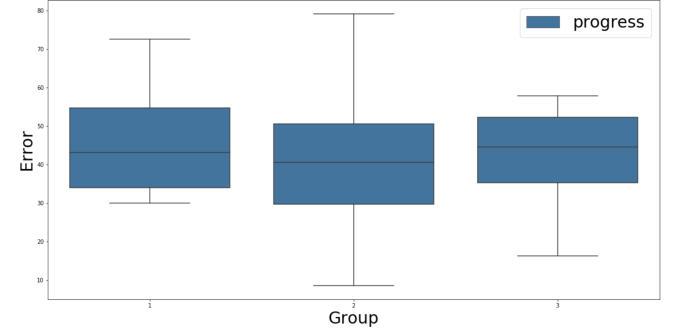
Fig. 14. Plots of the linear regression with two different initial gains and high speed. For each subplot, we have the pitch on the left and the roll on the right. The black dots represent initial point for the initial linear regression. Red dots are new values at waypoints. The black line is the initial gain (equal to initial gain) and the red line is the new gain after one runs. After one run, the sensibility adapts itself to 0.9 for the pitch and 1.2 for the roll. This seems my best gain for driving the drone.

The variation of the gains between each run is shown in fig. 16 and fig. 17.

After three-run, we have the same values between each phase (represent by 100 per cent). All these values converge to a similar range of gain for all the subjects. The range for the roll is [1.2,1.75] and for the pitch is [0.9,1.25]. We can see that on fig. 18



(a) Box plot of the mean error depending of the phase and the group.



(b) Box plot of the progress depending of the group.

Fig. 15. These two plots show us the error and the progress depending on the group. We cannot take conclusion on the improvement of the driving with or without the interface with these data. If the progress values have a low percentage, it shown that the user has well improved his drive.

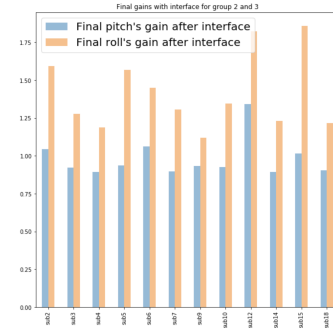
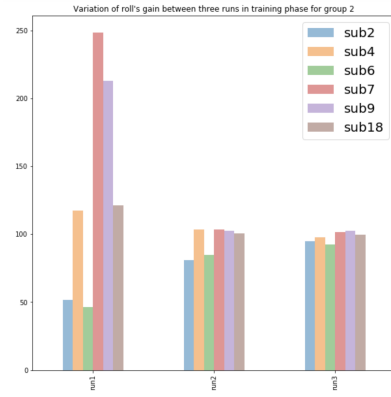


Fig. 18. Plot of the final gain with interface. All of the gains are not the same but have similar range. The range for the roll is [1.2,1.75] and for the pitch is [0.9,1.25]

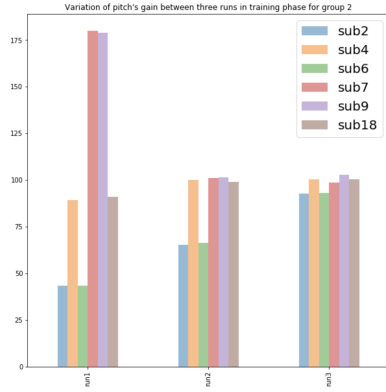
IV. DISCUSSION

The discussion is following the same conductor line as the results. First, we talk about the part that tries to maximize the similarity between an algorithmic spline and the human path. Then, we talk about the autopilot controller and the characteristic of the inputs. After, we talk about the linear regression. Finally, we discuss the experiment.

According to the results on the trajectory, not updating the path at every step is a way to create a smoother path. The



(a) Plot of the variation of the gain for roll between each run for group 2.



(b) Plot of the variation of the gain for pitch between each run for group 2.

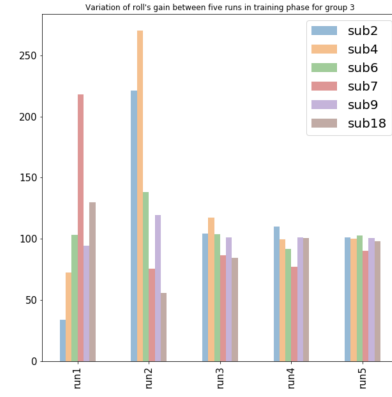
Fig. 16. These plots show the variation of the gains between each run for the group 2. 100 percent involves no change between two runs. After three runs the gain stop changing.

fact that updated the path at every step make the drone trajectory messy close to the waypoints. This phenomenon is due to Catmull-Rom spline did not work well if the points two points are close. And so, the inputs close to the waypoints are not similar to human inputs.

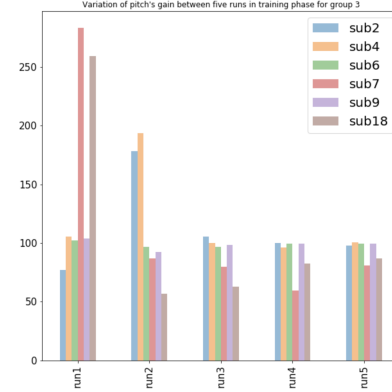
The new implementation of the spline maker with tension factor τ and the resolution has not brought the progress we hoped. One reason is that the waypoints are quite far from each other. The tension factor between 0-1 does not change the results sufficiently. We can underline that the best results are close to $\tau = 0.5$, showing us that humans try to minimize the slope.

To improve the similarity between human and mathematical spline, we can increase the number of waypoints and choose τ in a bigger range. if we have more waypoints, we have more control over the spline maker.

Concerning the autopilots controller, we notice that the PI inputs autopilots have the same shape as humans. We consider the mean of the human's inputs from the past study [Gregoire] to compare the shape. We can underline that the derivative term adds a lot of noise and instability. Thus, we



(a) Plot of the variation of the gain for roll between each run for group 3.



(b) Plot of the variation of the gain for pitch between each run for group 3.

Fig. 17. These plots show the variation of the gains between each run for the group 3. 100 percent involves no change between two runs. After three runs the gain stop changing.

preferred not to use it. It is also relevant that we cannot find a good autopilot that follows the curve only with a P controller. The different curvatures of the path are the reason. With a too big P gain, the drone will be noisy, and with a too-small P, the drone will not follow the path. More complex autopilots can be implemented to improve the path following. Nevertheless, the fact that PID inputs look like human inputs is the most important.

To smoothen the input and increase the similarity, we can add some complex filters inside the loop PID. For example, a PID with a controller output filter. It has not been used in this study because the similarity between PI controller inputs and human inputs are acceptable for our work.

flushleftThefirsthypothesiswemadetocombinehumanandPID, for

The subjects' experience does not give us all the hoped results. We try to prove that people learn faster with the interface and have more pleasure to drive with it. In a general case, the results tend in this direction. However, the experience of the user was too different from having a nice comparison.

To improve the experience's relevancy, we need to use more subject and harder difficulty in the game. For example, we can increase the speed in the game. More subjects will give us more information to generalize the data. Moreover, with the more challenging difficulty, the user will need more time to be used to the simulator, and the interface improvement can be perhaps seen.

About the characteristic of the interface, the converging gain for the iterative linear regression is encouraging. The interface converges not to one same gain for everybody. It converges after a few runs into the optimal gains for the user. Furthermore, it seems that people do not feel the change of sensibility, consequently during flight. Even is the gain having great change at the end of the run. Thus, the interface is doing its work without perturbing the drive of the user.

V. CONCLUSION

This report exposes an adaptive human-robot interface for aerial robot drive by teleoperation with iterative linear regression. It shows all the process to reach the goal. For instance, maximizing the similarity between the human trajectory and mathematical spline creates an autopilot with the same shape of inputs than humans and the implementation of the iterative linear regression. The results are encouraging. The interface converging efficiently to the user's preference.

In the future, we need to test the interface with more users, with different trajectories and different speeds to confirm that this method improves the drivers' accuracy and decreases the learning time.

bib