A Multimodal Control Architecture for Autonomous Unmanned Aerial Vehicles

Marco A. Gutiérrez
Robolab, University of Extremadura
Cáceres, Spain
marcog@unex.es

Luis Fernando D'Haro
HLT, I2R, A*STAR
Singapore
luisdhe@i2r.a-star.edu.sg

Rafael E. Banchs
HLT, I2R, A*STAR
Singapore
rembanchs@i2r.a-star.edu.sg

ABSTRACT

We present our preliminary work on a multimodal control architecture that enables an operator to manage an autonomous Unmanned Aerial Vehicle (UAV) through high level tasks in an indoors environment. The intelligence embedded in our architecture is able to decode these tasks into low level instructions that a UAV is able to execute. Our system allows the user to operate the UAV through speech, text or keyboard/mouse input, all presented in a web based graphical user interface that can be accessed from any Internet powered device.

ACM Classification Keywords

Robotics Operator interfaces

Author Keywords

Unmanned Aerial Vehicles; Natural Language Understanding; System Control; Visual Servoing;

INTRODUCTION

Research on autonomous UAVs has seen a huge growth along the last few years. Latest hardware advances along with the reduced costs of Micro-Aerial Vehicles (MAVs) have boost research in the field along with its applications. The range of UAVs applications and possibilities are currently very wide. They can be used for different purposes such as emergency management [4], wildlife monitoring [2], humanitarian relief actions [3] and much more. While for some of these solutions manual control is enough, others aim for more autonomous UAV solutions.

Our approach presents the controller with a web based interface that can be accessed from any device enabled with Internet and a web browser (computer, cell phone, tablet, etc.). The controller can communicate with the UAV through speech, text or using command buttons on the interface. The web interface also provides feedback through speech and visual elements like the UAV's camera along with tracking information. The user can command the UAV to autonomously perform a set of predefined high level tasks. Then the system architecture will take care of the UAV planing while providing the proper feedback to the controller so he can keep track of the correct execution of the task finally reaching the required goal.

HAI'16, October 4–7, 2016, Biopolis, Singapore.
ACM ISBN 978-1-4503-4508-8/16/10.
http://dx.doi.org/10.1145/2974804.2980522

DESCRIPTION OF THE ARCHITECTURE

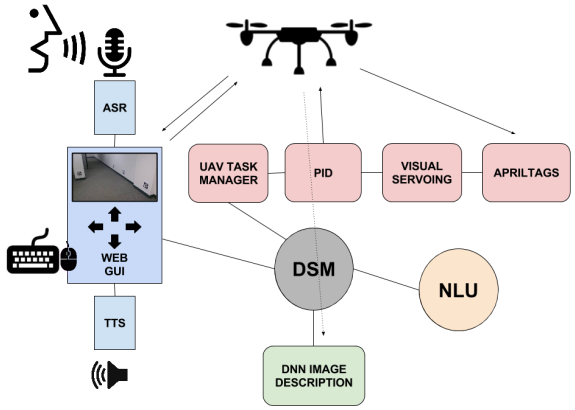

Figure 1. Drawing of the architecture of the system with its different modules. Draw of the architecture of the system with its different modules.

As stated, the architecture is composed of several modules implemented as one or more components of ROS. Following we explain more details on the implementation of each of these modules.

Web-based Graphical User Interface

The web based graphical user interface (see Figure 2) is used as the system's main point of communication with the end user. During regular usage, controller can interact with the system through natural language (either speech or text) to set the UAV task; however it also offers the option to override the high level tasks through specific buttons with direct access to the drone low level commands for emergency purposes. Specific information on the UAV task execution is also provided through different sensors information, camera image feedback (with tracking augmented information) and natural language.

The web interface is connected to ROS by using the rosbridge_suit package which uses JSON strings messages to communicate with ROS by means of a WebSocket connection between the browser and the ros server. In addition, it

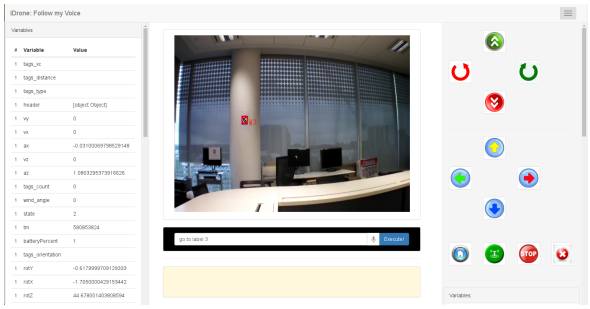

Figure 2. This is a screenshot of the web interface used to manage hte UAV. On the left you can see the sensors information while on the center the camera image and the ASR result is displayed, finally on left side is where the emergency-direct controls lay.

includes a JavaScript library that makes easy the creation and integration with services, publishers and subscribers available in ROS.

Finally, the interface communicates with the "ardrone_autonomy" ROS driver [6] which provides an interface to communicate with the official AR-Drone SDK version 2.0.1 through ROS. This way, the interface can obtain the UAV sensors' information (e.g. cameras' images, pressure, temperature, altitude, speed, etc.) as well as it is able to provide the user with the direct interface to override the high level task through UAV-specific basic commands for emergency purposes.

Dialog and System Manager

The Dialog and System Manager (DSM) is the module responsible of establishing the current state in the dialog flow while at the same time deciding what are the next high level actions the UAV must perform based on the current input from the user (and its interpretation given by the NLU module). This checking is inspired in the human autonomic nervous system (ANS), where critical tasks are performed independently, and with higher priority, over the somatic nervous system (SNS) in order to guarantee the correct functionality of the body. For instance, the ANS is responsible for controlling the heart movements, temperature, breathing, etc., while the SNS is responsible for the voluntary moments. In the robot, the SNS tasks are the ones defined for the developer to perform the typical tasks to be done when interacting with the human users, while the ANS tasks will be periodic checking of the robot internal functionalities (e.g. power status, temperature, close presence of humans, etc).

Examples of the tasks performed by the ANS module are the control of the battery level and altitude; in this case, when their current values trigger a predefined threshold the system automatically notifies, through the speech and web interface, to the controller in order to take the respective correction measures (e.g. returning the UAV to abort the task and go to a certain known and safe position).

Regarding the SNS tasks, the DSM module follows the states and transitions defined by a state machine where the designer must specify which are the different actions to be performed by the task manager in each state and the conditions to jump to a following state. In addition, the DSM is also responsible to classify some messages coming from different modules in the architecture and notify the user about the actions to be done based on them. For instance, if the confidence score of the speech recognition is below a given threshold then the DSM sends a message to the TTS to inform the controller that he needs to repeat or rephrase the utterance again. Another example is when the UAV needs to notify the controller that it reaches a certain position (i.e. detecting any of the markers).

Natural Language Understanding

The NLU module is responsible of providing an interpretation of a given sentence (e.g. a human sentence typed in on a textbox or recognized using an ASR) taking into account the active set of grammars and rules for the current state of the system.

It is based on the use of regular expressions that match a given sentence with a set of predefined patterns. The use of regular expressions provides a trade-off between accuracy, robustness, maintenance and available resources. In addition, the capability of the system to switch between different general and specific rules allows it to deal with the dynamic characteristics of the human-computer interactions.

This module consists of two main parts. First, a Python interface that is able to provide services and messages to other modules in the ROS-based framework. Then, a regular-expression based engine which allows the designer to match the possible set of natural language utterances spoken by the controller into an internal representation that extracts the most important concepts that can be obtained from the sentence and used by the DSM. In order to make the parsing and definition of the rules as flexible and robust as possible, the NLU handles three different XML grammar files: one that specifies the active grammars per state, another with the actual rules, and a third file containing a list of word mappings that can be used to reduce the verbosity level of the regular expressions (e.g. defining the set label NUMBERS and its items, and using this label to avoid writing all the possible numbers in the regular expression); in addition, this mapping could also be used to map specific words to internal codes or labels used for other modules in the architecture (e.g. replacing the words: up, above, roof, ceiling into a single internal word TOP). Figure 3 shows an reduced version of the regular expressions rules used by the UAV. The figure shows an example of a basic pattern (number 1), extracted slots (i.e. action, parameter, values) and more complex rules (number 2) using internal mappings (e.g. OBJECTS). Finally, this module also allows to atomize complex sentences into more simple sequential rules.

UAV Task Manager

The UAV Task Manager module is the one in charge of braking down the high level motion instructions coming from he DSM into the low level commands that the PID can understand and process to the UAV. Once the module gets a motion task from the DSM it breaks it and issues the proper targets to the PID module so the UAV can move towards its different positions to achieve the goal. The complexity of this module is not very

```
<!--Natural Language Understanding Grammar File -->
<nlu_grammars>
  <group name="root">
    <grammar name="grammars">
      <rule name="takeoff"></rule>
      <rule name="basic_commands">
        <!--
          <pattern> "start|stop|land|go|right|left|up|down" </pattern>
        -->
        <pattern>^(start|stop|land|go|right|left|up|down|rotate)$</pattern>  1
        <action>
          <frame>
            <slot name="action">basic_command</slot>
            <slot name="parameter">command</slot>
            <slot name="value">{1}</slot>
          </frame>
        </action>
      </rule>
      <rule name="finish"></rule>
      <rule name="goto_labelx"></rule>
      <rule name="goto_xlabel"></rule>
      <rule name="come_back"></rule>
      <rule name="find_something">
        <!--<pattern> "find a OBJECT" </pattern>-->
        <pattern>
          (.*)?(find|search|look for)\s*(a|the)?\s+({OBJECTS})(\s+{PLEASE})?$  2
        </pattern>
        <action>
          <frame>
            <slot name="action">find_one_object</slot>
            <slot name="parameter">object_1</slot>
            <slot name="value">{4}</slot>
          </frame>
        </action>
      </rule>
      <rule name="find_two_things"></rule>
      <rule name="take_picture"></rule>
      <rule name="take_picture_zoom"></rule>
      <rule name="tell_me_what_you_see"></rule>
      ■■■
    </grammar>
  </group>
</nlu_grammars>
```

Figure 3. Example of XML rules used by the NLU.

high but it is expected to grow as the motion tasks requested to the UAV become more complicated in future developments.

Visual Servoing with markers

When localizing UAVs in GPS-denied environments, such as indoors setups, external motion capture systems are usually used, i.e. indoors ball catching [5]. Although there are other solutions such as [1], this relies on the existence of texture in the surroundings. Therefore we decided to create a markers based visual servoing system to localize the UAV in our indoors environment since this would provide more robustness and simplicity to the system.

This module uses a ros node that tracks the markers previously set up in known locations through the scene. This position is then received by the visual servoing node that computes the location of the UAV within the indoors location. One drawback of our system is that when the UAV does not see any marker it cannot locate itself within the room, therefore it enters a "tag-search mode", where it rotates over the yaw angle, ψ until it finds a new marker and its able to locate itself again. Since, in our setup, there is at least one marker on each wall the UAV is able to find a new marker in a reasonable amount of time, therefore the time it is lost is actually very small. Obviously the more markers you have in the setup the less likely the UAV will be to get lost and the easiest for it to relocate itself in the unlikely case that this happens.

Finally on this module an Extended Kalman Filter (EKF) in order to make the markers detection more stable in time. We had to add this improvement when we moved our tests to real environments as the markers detection was not as constant as in the virtual scenario.

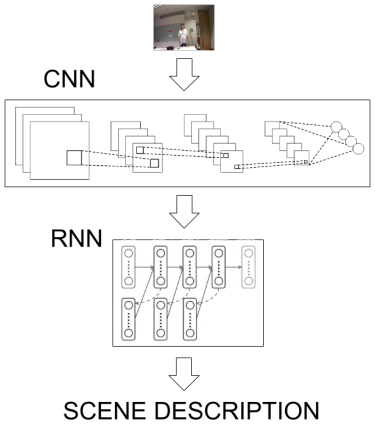

Figure 4. Structure of the DNN used by the UAV to describe the scene images.

PID

The PID controller is the one in charge of generating the control signals that are sent to the UAV through the "ardrone_autonomy" driver at a rate of 100Hz. We used a similar approach of the controller described in [1]. The control signals define the proper roll $\bar{\phi}$ and pitch $\bar{\theta}$ angle, the yaw rotational speed $\bar{\psi}$ and the vertical velocity $\bar{z}$. All of them are defined as a fraction of the values: 18° for roll and pitch, 90° for yaw speed and $2m/s$ for vertical velocity.

The PID will receive the UAV current position from the Visual Servoing module $p_t = (x, y, z, \phi, \theta, \psi)$ and when a target position is set $p_t = (\breve{x}, \breve{y}, \breve{z}, \breve{\psi})$ by the UAV Task Manager, it will apply a separate PID control to all four degrees of freedom. The result is rotated to match the yaw orientation of the UAV. The control gains where optimized experimentally and are set as follows:

$$\begin{pmatrix} \bar{\phi} \\ \bar{\theta} \\ \bar{\psi} \\ \bar{z} \end{pmatrix} = \begin{pmatrix} P_r(\psi) \begin{bmatrix} 0.5(\breve{x}-x)+0.32\dot{x}) \\ 0.5(\breve{y}-y)+0.32\dot{y}) \end{bmatrix} \\ 0.02(\breve{\phi}-\phi]) \\ 0.6(\breve{z}-z)+0.2\dot{z}+0.01\int(\breve{z}-z) \end{pmatrix} \quad (1)$$

Where the current speed is denoted by $v_c = (\dot{x}, \dot{y}, \dot{z})$ and $P_r(\psi)$ denotes a planar rotation by ψ.

Environment Description Module

The environment description module is the one in charge of describing the UAV scenes. As user demand, the UAV will describe the scene that it's in front of him through the Text to Speech (TTS) system. This could be a useful feature for visually impaired people as they can get a description of what is in front of he UAV without the need to actually see the image.

The flow of scene description generation is as follows. When a users makes a scene description request through the web interface and the DSM identifies the task it requests the DNN Image Description module for a description of the current

scene. Then the image is captured, the description is generated and read to the user through the TTS system.

To generate this scene explanations the DNN image description module implements the Neural Image Caption model (NIC) described in [7]. As figure 4 shows, this model uses a Convolutional Neural Network (CNN) pre-trained for image classification as image encoder. Then the last hidden layer is of this network is used as an input to a Recurrent Neural Network (RNN) decoder that generates the sentences.

CONCLUSIONS AND FUTURE WORK

We have described our preliminary work on a multimodal control architecture for autonomous UAVs. Our architecture provides a solid base to develop high level tasks on autonomous UAVs. Our system is be ale to locate the drone and execute human commanded autonomous tasks.

However since we would like to make our system suitable for real applications, further work and testing would be needed to achieve more complicated and robust high level tasks executions. We would like to have a more robust localization system in order to avoid as much as we can the “getting lost" situations. Also removing the markers will open our system to higher and easier mobility. Since accurate indoors localization for these systems remains an open challenge, this would require further research and efforts trying out other possible technologies.

Finally we would also like to test our architecture and the future enhancements with more and improved UAVs as the limitations imposed by the hardware in the current platform restricts the possibilities of our applications.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Jetson TX1 used for this research.

REFERENCES

1. Jakob Engel, JÃijrgen Sturm, and Daniel Cremers. 2014. Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems* 62, 11 (2014), 1646 – 1656. DOI: http://dx.doi.org/10.1016/j.robot.2014.03.012 Special Issue on Visual Control of Mobile Robots.

2. Luis F. Gonzalez, Glen A. Montes, Eduard Puig, Sandra Johnson, Kerrie Mengersen, and Kevin J. Gaston. 2016. Unmanned Aerial Vehicles (UAVs) and Artificial Intelligence Revolutionizing Wildlife Monitoring and Conservation. *Sensors* 16, 1 (2016), 97. DOI: http://dx.doi.org/10.3390/s16010097

3. M. A. Gutiérrez, S. Nair, R. E. Banchs, L. F. D. Enriquez, A. I. Niculescu, and A. Vijayalingam. 2015. Multi-robot collaborative platforms for humanitarian relief actions. In *Humanitarian Technology Conference (R10-HTC), 2015 IEEE Region 10*. 1–6. DOI: http://dx.doi.org/10.1109/R10-HTC.2015.7391867

4. Jinjun Rao, Zhenbang Gong, Jun Luo, and Shaorong Xie. 2005. Unmanned airships for emergency management. In *IEEE International Safety, Security and Rescue Rototics, Workshop, 2005.* 125–130. DOI: http://dx.doi.org/10.1109/SSRR.2005.1501243

5. R. Ritz, M. W. MÃijller, M. Hehn, and R. D’Andrea. 2012. Cooperative quadrocopter ball throwing and catching. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 4972–4978. DOI: http://dx.doi.org/10.1109/IROS.2012.6385963

6. Autonomy Lab Simon Fraser University. 2016. ardrone_autonomy Parrot AR-Drone 1.0 and 2.0 quadrocopter ROS driver. (2016). https://ardrone-autonomy.readthedocs.io/en/latest/index.html.

7. Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2014. Show and Tell: A Neural Image Caption Generator. *CoRR* abs/1411.4555 (2014). http://arxiv.org/abs/1411.4555