2015 IEEE/SICE International Symposium on System Integration (SII)
December 11-13, 2015. Meijo University, Nagoya, Japan

Control Architecture for Service Drone in Informationally Structured Environment

Farouk Ghallabi, Akihiro Kawamura, Yoonseok Pyo, Tokuo Tsuji, and Ryo Kurazume

Abstract— **In this paper, we present a control architecture that enables a service drone to navigate in an informationally structured environment (ISE) and to accomplish a specific service task autonomously based on the ROS-TMS framework. The ROS-TMS is a ROS-based distributed information management system for ISE. It manages a variety of subsystems ranging from sensing by distributed sensors to motion planning and behavior control for service robots. The proposed architecture is designed as a component of the ROS-TMS, and consists of a navigation system that solves a path planning in ISE and a flight control system by a behavior-based finite state machine. The navigation of the drone is allowed by an optical motion tracking system consisting of distributed infrared cameras managed by the ROS-TMS. A graphical user interface is designed to provide simple manipulation of a service task by a drone. Service experiments have been conducted to validate the performance of the architecture.**

I. INTRODUCTION

Unmanned aerial vehicle (UAV), generally known as a drone, is used in many situations where manned flight is considered to be difficult, too risky, or in a few cases impossible. In recent years, the use of these unmanned aircrafts, especially quadcopters has become widespread in many fields as they possess great locomotive abilities and have proved very efficient in performing tasks. Such tasks include mainly civil missions, for example the traffic surveillance [1], [2], the monitoring of oil and gas pipelines and power lines [3], missions of civil security such as the people detection [4]. Many other uses are being developed in various fields like agricultural spraying [5], vegetation monitoring [6], region mapping, promotional films for tourism and sports, ecological surveys, fire extinguishing, etc. Hence, these flying robots are the subject of continuous labor and research in order to provide drones that can carry out their missions with the least fail probability as possible.

The aim of this research is to develop an application using a parrot platform for human daily life assistance in an informationally structured environment (ISE). The ISE is an intelligent environment where a number of sensors are embedded. Various information such as events occurred in the environment, positions and situation of objects, humans, and robots are sensed and stored structurally in a database. We have been developing the ROS-TMS framework [7] in ISE, which is a ROS-based distributed information management system. The ROS-TMS manages a variety of subsystems ranging from sensing by distributed sensors to motion planning, and behavior control for service robots. The control architecture for a service drone proposed in this paper is designed as a component of the ROS-TMS. The architecture consists of a navigation system that solves a path planning problem in ISE, a non-linear flight control system for a drone, and a behavior control system by a behavior-based finite state machine. In addition, a graphical user interface is designed to provide simple manipulation of the drone.

The rest of the paper is organized as follows. After introducing the concept of ISE and ROS-TMS in Section II, we will explain the drone platform in Section III. In Sections IV and V, the control system for a drone consisting of a path planning system and non-linear flight control system will be introduced. In Section VI, we will show the graphical interface for making a service task by a drone. Finally, experimental results in a room will be shown in Section VII.

II. Informationally Structured Environment and ROS-TMS

A. Informationally Structured Environment

Many researches to design autonomous systems have been conducted in robotic field. The basic idea is to develop robots that take decisions without human assistance. Doing such a work is not trivial and requires some conditions to be satisfied. In fact, robots cannot execute tasks autonomously without interacting with the physical world. Therefore, the information about the environment is of key importance for designing autonomous systems. Various sensor devices have been designed to sense some characteristics of the physical world. However, until today, no single sensor is able to provide all of the information about the environment. Thus, robot platforms are equipped with many sensors to achieve tasks. For instance, global positioning system (GPS) is used to localize a robot in the 3D world, inertial measurement units are used to sense accelerations and angular velocities, wheel encoders are used to measure the speed or distance of a wheel travels, etc. Collecting all kinds of sensors on robots is obviously not achievable due to the hardware capacity and resource limitation. An informationally structured environment (ISE) is a concept that supports robot's knowledge by offering different sensor modalities distributed in the world. In the ISE, a number of sensors are embedded not only

Farouk Ghallabi is with ENSTA ParisTech, 1024, Boulevard des Maréchaux, 91762 Palaiseau Cedex, France farouk.ghallabi@ensta-paristech.fr

A. Kawamura, T. Tsuji and R. Kurazume are with Faculty of Information Science and Electrical Engineering, Kyushu University, 744, Motooka, Nishi-ku, Fukuoka, Japan {kawamura,tsuji, kurazume}@ait.kyushu-u.ac.jp

Y. Pyo is with Graduate School of Information Science and Electrical Engineering, Kyushu University, 744, Motooka, Nishi-ku, Fukuoka, Japan pyo@irvs.ait.kyushu-u.ac.jp

on a robot but also in an environment. Additionally, sensed information is stored structurally in a database. If a robot requests an information to the database, the information is sent back to the robot immediately.

As an example of the ISE, we have been developing an intelligent room named Big Sensor Box (B-sen) as shown in Fig. 1. In B-sen, various sensors are embedded such as laser scanners, RGB-D cameras, intelligent cabinets, intelligent refrigerator, floor sensing system, etc. For tracking positions of human, robots, and objects, Vicon motion capture system is installed. This sensor detects positions of human, robots, and objects in B-sen accurately. Vicon system detects objects through a passive optical motion capture technique, which uses retroreflective markers tracked by infrared cameras as illustrated in Fig. 2. Therefore, in this study, many markers are placed on a drone, as illustrated in Fig. 3, so that we can get the orientation and position information of the drone in B-sen. The Vicon system has been used for team motion control of multiple quadcopters [8], [9], [10]. However, in these systems, the frame rate of the position data is quite high. It is 100∼375 [Hz]. In our system, since the vicon system is one of the embedded sensors in ISE, such a high bandwidth cannot be assigned to the vicon system due to the limitation of the available bandwidth. In our system, the frame rate of the vicon data is about 10∼100 [Hz] and it causes unstable behaviors for the drone in some cases.

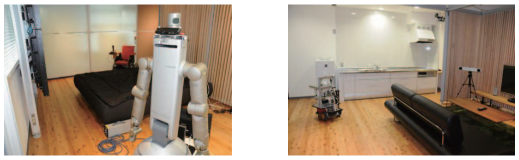

Fig. 1. An informationally structured environment named Big Sensor Box (B-sen)

Fig. 2. Vicon motion capture system (Bonita) and a retroreflective marker

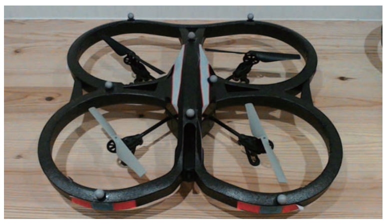

Fig. 3. Drone covered by retroreflective markers

The information gathered by the sensors including Vicon is processed and sent to a centralized management system [11] which, in turn, broadcasts it via a local network to all available robots. The management system uses ROS to manage data and to publish it. The software developed is called ROS_TMS (Town Management System). In the next subsection, ROS_TMS architecture is briefly described.

B. ROS_TMS architecture

ROS_TMS [7] is a software designed to manage information received from sensors distributed in ISE. The data is stored in a local database, processed according to each task, and delivered to all robot services. Data management, processing, and scheduling are executed using more than 90 ROS modules, as listed below and illustrated in Fig. 4:

- The user request (TMS_UR) is for communicating with the user by using a remote controllers, smartphones, tablets, and other portable devices, interpreting the commands sent by the user and delivering it to the task scheduler.
- The task scheduler (TMS_TS) receives the commands from the task request and manages the executions of these commands.
- Sensor and robot controller modules (TMS_SD and TMS_RC) are responsible for interpreting the data from sensors and computing the commands to execute each task.

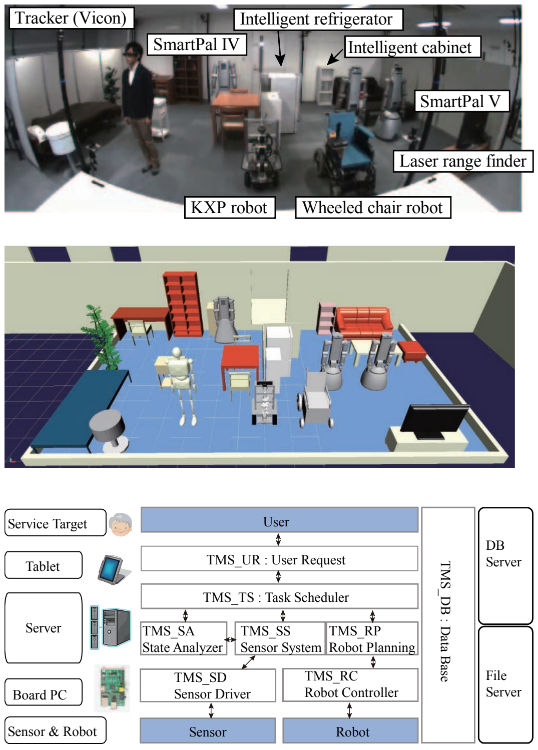

Fig. 4. ROS_TMS architecture

As ROS_TMS is based on ROS, data broadcasting is done by ROS publisher nodes. For instance, in order to get Vicon data, we need to subscribe to a specific ROS node. This node reads data from the database and publishes it through a local network.

III. Parrot Ar.drone Platform

We have been working on Parrot Ar.drone 2.0 platform shown in Fig. 5. This platform has 3-axis gyroscope and accelerometer, ultrasound sensor, two cameras: the front camera with high resolution used for recording images and videos, the bottom camera with lower resolution and used to estimate the altitude of the drone.

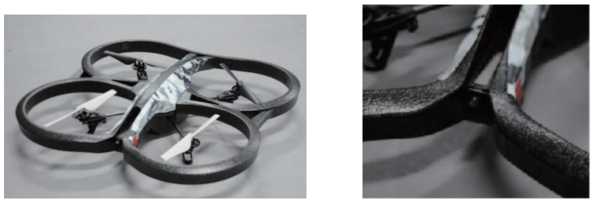

Fig. 5. Parrot Ardrone 2.0 and front camera

Parrot platform is also equipped with a cortex A8 microprocessor, and this processing unit uses the onboard sensors to stabilize the system, to maintain a fixed altitude, to control the pitch, roll and yaw angles. Parrot is controlled by sending four commands $u = (u_x, u_y, u_z, u_\psi)$, and all of them should be in $[-1, 1]$. The (u_x, u_y) commands control the horizontal speeds in the x and y directions respectively. The u_z command controls the vertical speed, and the u_ψ controls the angular velocity around the vertical axes z. As we mentioned above, the microprocessor executes an inner loop that controls the pitch, roll, and yaw angles. Thus the control commands are simply percentages of maximum values of these parameters, positive and negative values determine the direction of the movement. To control the drone, Wifi connection must be established and specific commands must be sent. As we use ROS, we have been using ardrone_autonomy [12].

IV. Map Representation and Path Planning

As we mentioned in the introduction, the system has to solve a path planning problem in ISE and control a drone motion to track the generated path. Thus, a dynamic model is of particular value to understand the drone behavior and to design a robust controller. The following sections will deal with these problems and will demonstrate the methods used to solve them.

A. Occupancy grid map

The goal of this part is to solve a path planning problem in ISE. In ISE, all of the necessary information including the structure and positions of obstacles are obtained beforehand. Thus this is of key importance for designing autonomous systems and for finding shortest, or optimal, path between two points. In this study, we plan to develop a path planning algorithm as follows:

1) Localize the drone in the environment
2) Input a goal position by GUI device
3) Compute the shortest path between initial and goal positions
4) Compute a safe path (avoiding existing obstacles).

To do that, we need to represent the environment. Starting from a 3D model, as shown in Fig. 6, we created a 2D occupancy grid map in which only xy plane is considered to simplify the planning problem. Ideally, since the drone lives in a 3D world, a 3D map should be considered in the future.

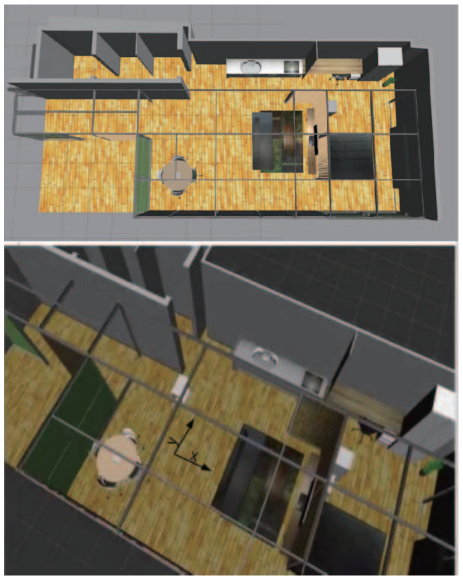

Fig. 6. 3D model of the environment

Since the indoor hull of the drone is about $517mm \times 451mm$ [13] and the dimension of the environment is $12.5m \times 4.5m$, we selected 0.5m as the resolution of the grid map. Therefore, the occupancy grid map consists of 25×9 grids. In practice, to insure safe flights, some virtual obstacles are added, for instance, this is convenient to fill narrow places and unreachable regions for the drone and to insure a safe distance from true obstacles.

Finally, for completion of the work, the drone position is read from Vicon sensors and marked on the map. Figure 7 shows a graphical display of the map using Matlab. White squares refer to obstacle-free spaces, black ones refer to obstacle spaces, and the gray one is the drone position.

B. Dijkstra's algorithm

Now that we defined a 2D map, a Dijkstra's algorithm is implemented to find a shortest path between two positions. Dijkstra's algorithm uses a weighted graph $G = (E, V)$. The set V represents the set of vertices of the graph G, and the set E is the set of couples $(u, v) \in V^2$. For each couple (u, v), we define a cost function $w(u, v)$, which must be nonnegative, i.e: $w(u, v) \geqslant 0$.

In our case, the map elements are labeled from 1 to 255 (25×9). The set V is constructed as

$$V = \{i \in 1, 2, .., 255, \text{such that element } i \text{ of the map is free(no obstacle)}\}, \quad (1)$$

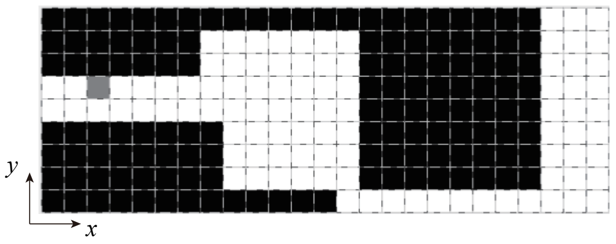

Fig. 7. Occupancy grid map

where V is the set of obstacle-free spaces in the occupancy grid map. Now, we need to construct the set E. Our approach consists in reducing the movements taken by the drone into four elementary movements: {forward (F), backward (B), left (L), right (R)}, that means, from a given vertex, maximum four vertices are possible to be selected.

The last thing to do is defining the cost function w. Let u and v be two adjacent vertices. Suppose that we are currently in vertex u then

$$w(u,v) = \begin{cases} 1, & \text{if } v \text{ is reachable} \\ \infty, & \text{if } v \text{ is not reachable} \end{cases} \tag{2}$$

V. Behavior control by Finite State Machine

A. Quadcopter dynamics

The drone is a rigid body that lives in a six degrees of freedom parametrized by a translation and orientation vectors. The translation vector can be identified as the position of its center of mass $\mathbf{x} = (x, y, z)$ and the orientation vector can be parametrized according to roll-pitch-yaw convention (θ, ϕ, ψ). Forces applied on the system are generated by four motors and propellers, these forces induce three torques (τ_x, τ_y, τ_z), around three body axes as shown in Fig. 8.

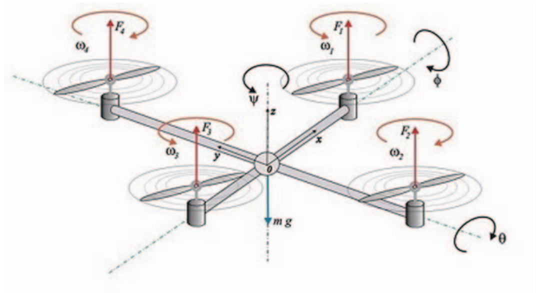

Fig. 8. Quadcopter parameters

Let's denote by $\mathbf{f}$ the total thrust generated by the four propellers, R_{RPY} the rotation matrix, and m the drone mass. According to roll-pitch-yaw convention one representation of the matrix R_{RPY} is

$$R_{RPY} = \begin{pmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - S_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{pmatrix} \tag{3}$$

For simplicity, we denote the terms $\cos\theta$ and $\sin\theta$ by C_θ and S_θ, respectively.

1) Dynamics: Acceleration of the center of mass can be derived from Newton's law

$$\ddot{\mathbf{x}} = (R_{RPY}\mathbf{f} - \mathbf{f}_g)/m \tag{4}$$

Let $\mathbf{f} = (0,0,f)^T$ and $\mathbf{f}_g = (0,0,-g)^T$, the equations of motion are:

$$\ddot{x} = \frac{C_\phi S_\theta C_\psi + S_\phi S_\psi}{m} f \tag{5}$$

$$\ddot{y} = \frac{C_\phi S_\theta S_\psi - S_\phi C_\psi}{m} f \tag{6}$$

$$\ddot{z} = \frac{C_\phi C_\theta}{m} f - g \tag{7}$$

Let's make a simple interpretation of these equations and point out some remarks. The generated thrust is the sum of the forces produced by the system {motor+propeller}. In fluid mechanics, the force generated by a propeller is proportional to the square of the rotational speed of each motor. Therefore, the values of motor speeds are required to know f. However, in some platforms, like parrot ar.drone, we do not have access to this information. As a consequence, the value of f is considered as an unknown function. This said, using the equations of motion may not be simple and requires knowledge about f or about its estimate. Our approach is planned to solve this problem and to simplify the 3D dynamic model so that we can design a suitable feedback controller.

B. Behavior based control approach

In order to simplify the dynamic model expressed in the previous subsection, behaviors have been implemented to reduce the degrees of freedom and to make the control design easy. Before starting demonstrating the method, it is convenient to start by defining the meaning of behaviors in robotics. Ronald C.Arkin defines in his book [14] a behavior as a *reaction to a stimulus*. Stimulus has different forms, for instance if we consider a mobile robot navigating in an unknown environment one behavior can be 'avoid-obstacle'. This behavior is not active until an obstacle is detected, hence "obstacle-detected?" can be seen as a stimulus to "avoid-obstacle" behavior. Many approaches have been used to model behaviors. Ronald mentioned in the same book three approaches to express behaviors.

1) Stimulus-response (SR) diagram
2) Functional notation
3) Finite state acceptor (FSA) diagram

Our case study uses the third approach to express the behaviors implemented for parrot platform. *"FSAs are best used to specify complex behavioral control systems where entire sets of primitive behaviors are swapped in and out of execution during the accomplishment of some high-level goal"* [14]. It is represented by a set of behavior states and transitions between them. A transition can be expressed by {condition+action}. Figure 9 shows a simple FSA with two states and one transition.

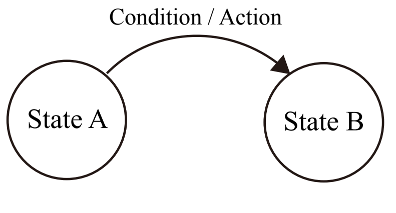

Fig. 9. Finite state acceptor diagram

1) Model Simplification: In this part, we will describe steps which have been taken to simplify the model. Indeed, in the Path Planning section, the generated path is a simple set of discrete 2D points (x_{ref}, y_{ref}). This does not include the altitude of the drone z and suppose that it moves only in a 2D world. Moreover, orientation is not expressed so that for a fixed (x_{ref}, y_{ref}) we have infinite values of $\psi \in [0, 2\pi]$. This suppose that during trajectory tracking, ψ and z have fixed values which is not always satisfied. That's why to keep validating these assumptions. The behaviors are implemented to control orientation and altitude of the drone as illustrated in the following points:

1) **Controlling the orientation (yawControl)**
   Consider the room model showed in Fig. 7, two major directions can be identified.
   - The drone is moving in the x-positive direction so that $\psi = 0$
   - The drone is moving in the x-negative direction so that $\psi = \pi$

   The yawControl behavior uses a proportional controller to control the orientation. Thus, the control command sent to the drone is $u_\psi = K_1(\psi - \psi_{ref})$ where $\psi_{ref} = \{0, \pi\}$
2) **Controlling the altitude (altitudeControl)**
   This behavior is for controlling the altitude at a fixed height $z = z_{ref}$. This behavior also uses a proportional controller to control the altitude. The command sent is $u_z = K_2(z - z_{ref})$
3) **2D path tracking (pathTracking)** This behavior is for tracking the reference points (x_{ref}, y_{ref}) generated by the path planning algorithm. The control technique used in this behavior will be discussed in the next part.

If we consider that during the 2D path tracking, the altitude remains constant, i.e $z = z_{ref}$, vertical acceleration can be set to 0 and then we can compute the value f. Conforming to (7) and to the result $\ddot{z} = 0$, the expression of f is deduced as $f = \frac{mg}{C_\theta C_\phi}$. Using the new expression of f and the possible values of ψ we retrieve the following systems:

$$\ddot{x} = \pm g \tan(\theta) \quad (\psi = 0 \text{ or } \pi) \tag{8}$$

$$\ddot{y} = \mp g \frac{\tan(\phi)}{\cos(\theta)} \quad (\psi = 0 \text{ or } \pi) \tag{9}$$

$$\ddot{z} = 0 \tag{10}$$

One extra behavior can be added for the parrot platform which is hover mode. In fact, hover command enables the drone to maintain a fixed position (x, y, z), this is convenient to insure safety when the previous behaviors are swapped in and out during execution.

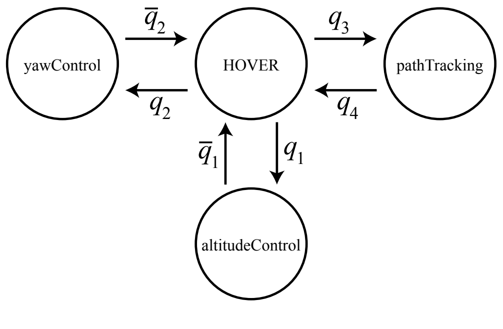

Fig. 10. Finite State Acceptor diagram

Figure 10 illustrates behaviors and transitions implemented as described in this section. As it was mentioned, transition can be viewed as simple condition q, the notation $\bar{q}$ refers simply to the negative of condition q. Transitions are described as follows:

- $q_1 = \|z - z_{ref}\| > \epsilon_1$
- $q_2 = \|\psi - \psi_{ref}\| > \epsilon_2$ and $\|z - z_{ref}\| < \epsilon_1$
- $q_3 = \bar{q}_1$, $\bar{q}_2$ and $\|(x, y) - (x_{ref}, y_{ref})\| > \epsilon_3$
- $q_4 = \|(x, y) - (x_{ref}, y_{ref})\| < \epsilon_3$,

where ϵ_1, ϵ_2 and ϵ_3 are positive values fixed throughout experiments. We used $\epsilon_1 = 0.2$ [m], $\epsilon_2 = 0.2$ [deg] and $\epsilon_3 = 0.4$ [m] in the experiments, respectively. Note that as the step value for the grid map described in the previous section is 0.5 m, it is convenient to have $\epsilon_3 < 0.5$, otherwise we will not have satisfying results. Indeed, take the example of $\epsilon_3 = 1$, the pathTracking behavior is inactive when the condition q_4 is satisfied, or $\epsilon_3 = 1$ is like allowing the tracking error to be represented by two squares in the map, hence the goal position might not be tracked in the desired square. Moreover, transitions are defined in a manner that there is a predefined execution order. For instance, condition q_2 cannot be true until condition $\bar{q}_1$ is true. The execution orders are

1) altitudeControl
2) yawControl
3) pathTracking

The implementation of this diagram has been computed using Stateflow simulink toolbox.

C. 2D position control

Now that we simplified the dynamic model using behavior-based design, let's get down to the details of the pathTracking behavior. We saw that the pathTracking behavior is intended to provide a control approach for tracking 2D paths generated by the path planning algorithm. In this part, (8) and (9) in case $(\psi = 0)$ are considered, note that the same work is done for the case $(\psi = \pi)$.

We want the drone to track a reference points (x_{ref}, y_{ref}). As we mentioned in Section 2, the input commands for parrot platform are percentages of certain maximum values for these angles.Thus, the input commands are $(\theta_{des}, \phi_{des})$, where θ_{des} is the desired angle for roll, and ϕ_{des} is the desired angle for pitch. Equations (8) and (9) can be written as

$$\begin{aligned} \ddot{x} &= g\tan(\theta_{des}), & (11)\\ \ddot{y} &= -g\tfrac{\tan(\phi_{des})}{\cos(\theta_{des})}. & (12) \end{aligned}$$

Suppose that we want to design θ_{des} and ϕ_{des} in a way that we transform the 2D dynamics equations ((11) and (12)) to a second order system defined by

$$\ddot{x} = \alpha_1\dot{x} + \beta_1 x, \quad (13)$$
$$\ddot{y} = \alpha_2\dot{y} + \beta_2 y. \quad (14)$$

This system can be written in the state-space form $\dot{\mathbf{x}} = A\mathbf{x}$, where $\mathbf{x} = (x, \dot{x}, y, \dot{y})^T$ and

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \beta_1 & \alpha_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \beta_2 & \alpha_2 \end{pmatrix}. \quad (15)$$

The system is asymptotically stable only if the eigenvalues of A have strict negative real parts. Equations (11), (12), (13), and (14) yield the following system:

$$g\tan(\theta_{des}) = \alpha_1\dot{x} + \beta_1 x \quad (16)$$

$$-g\frac{\tan(\phi_{des})}{\cos(\theta_{des})} = \alpha_2\dot{y} + \beta_2 y \quad (17)$$

$$-g\tan(\phi_{des}) = \cos(\theta_{des})(\alpha_2\dot{y} + \beta_2 y) \quad (18)$$

$$-g\tan(\phi_{des}) = \cos\left(\arctan\left(\frac{\alpha_1\dot{x} + \beta_1 x}{g}\right)\right)(\alpha_2\dot{y} + \beta_2 y). \quad (19)$$

The control inputs sent to the drone are

$$\theta_{des} = \arctan\left(\frac{\alpha_1\dot{x} + \beta_1 x}{g}\right) \quad (20)$$

$$\phi_{des} = -\arctan\left(\frac{\cos(\arctan(\frac{\alpha_1\dot{x}+\beta_1 x}{g}))(\alpha_2\dot{y} + \beta_2 y)}{g}\right). \quad (21)$$

Note that we set $\dot{x} = \dot{x}_{ref} - \dot{\tilde{x}}$, $\dot{y} = \dot{y}_{ref} - \dot{\tilde{y}}$, $x = x_{ref} - \tilde{x}$, and $y = y_{ref} - \tilde{y}$ in (20) and (21). $\dot{\tilde{x}}$, $\dot{\tilde{y}}$,$\tilde{x}$, and $\tilde{y}$ are current velocities and positions obtained by Kalman filter in the next section, and $\dot{x}_{ref}$, $\dot{y}_{ref}$, x_{ref}, and y_{ref} are target velocities ($\dot{x}_{ref} = \dot{y}_{ref} = 0$) and next reference positions.

Let's point out some remarks. As it has been mentioned in section 2, control commands must be in $[-1, 1]$ or (20) and (21) do not guarantee that we can satisfy such a condition. Three steps are taken to get rid of this problem. First, in the choice of the coefficients α_1, α_2 and β_1, β_2, for that reason, simulations are running to pick out acceptable coefficients. Second, as the control commands are percentages of ϕ_{max} and θ_{max}, desired angles θ_{des} and ϕ_{des} are divided by θ_{max} and ϕ_{max}. Finally, saturation functions are added to eliminate unexpected overshoots.

1) Outer loop/Inner loop: The onboard software uses these commands (20) and (21) to control roll and pitch. As we do not know exactly what algorithm is running inside the system, many experiments have been conducted to estimate the control parameters running behind the scenes. In [15], the influence of control commands as a first order linear model is described so that the real pitch and roll angles as the following equations:

$$\begin{aligned} \dot{\theta} + \lambda_1\theta &= c_1\theta_{des}, & (22)\\ \dot{\phi} + \lambda_2\theta &= c_2\phi_{des}. & (23) \end{aligned}$$

In order to estimate λ_1 and λ_2, step commands are sent to the drone, i.e: a set of fixed values of θ_{des} and ϕ_{des}, angular measurements are collected from the onboard sensors for few seconds. Figure 11 shows an example of an angular response for $\theta_{des} = \alpha\theta_{max}$, where $\alpha = 0.4$ and $\theta_{max} = 0.21$ [rad]. The shape of the curve validates approximating the response as a first order system. Let's write (22) and (23) in the Laplace domain:

$$\begin{aligned} \theta(s) &= \frac{c_1}{s + \lambda_1}\theta_{des}, & (24)\\ \phi(s) &= \frac{c_2}{s + \lambda_2}\phi_{des}. & (25) \end{aligned}$$

Due to the symmetric geometry of the drone we can assume that roll and pitch follow the same response so that we can assume that $\lambda_1 = \lambda_2 = \lambda$ and $c_1 = c_2 = c$.

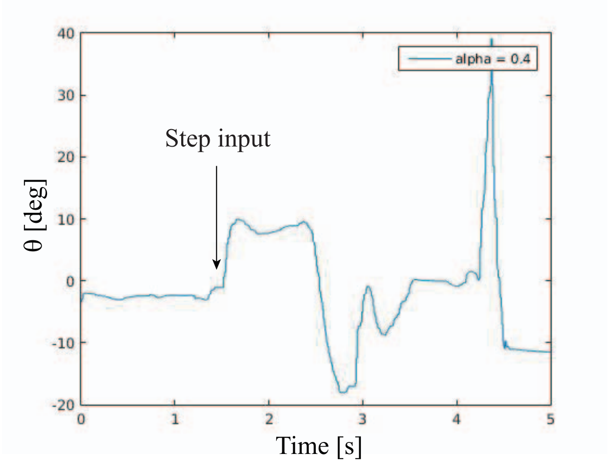

Fig. 11. Examples of roll responses

Using these data in System Identification toolbox in Matlab enables as to have an estimation of (c, λ). These parameters are of particular value for the state estimation and Kalman filtering process in the next part.

Now that we determined (c, λ), two independent loops are running. The outer loop is going to compute the desired angles according to the desired position as described above. The onboard loop, the inner loop, is going to stabilize the system around these commands. Figure 12 resumes the architecture of Outer/Inner loops.

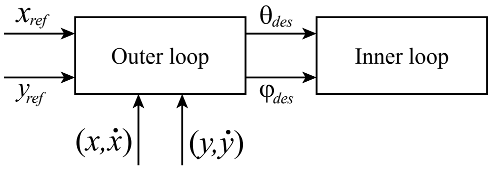

Fig. 12. Inner/outer loops

D. State estimation and Kalman filter

In the previous part, (20) and (21) are computed using $(x, \dot{x}, y, \dot{y})$. This implicitly supposes that information on position and velocity is available and with no errors. In practice, none of these two assumptions is true. Vicon data only provides position and orientation. Moreover, there is no guarantee to have a noiseless data. That's said, a state estimation and filtering techniques are required for good computation of feedback control. A Kalman filter is a good estimator for our case, and thus we adopted the Kalman filter for state estimation. Control System toolbox in Matlab provides a simple Simulink block to implement it. This block requires measurement data and input commands and outputs the estimate of x which we will be used for the control techniques discussed in the previous parts. Figure 13 illustrates a completion of Fig. 12 integrating Kalman filter and Vicon_data.

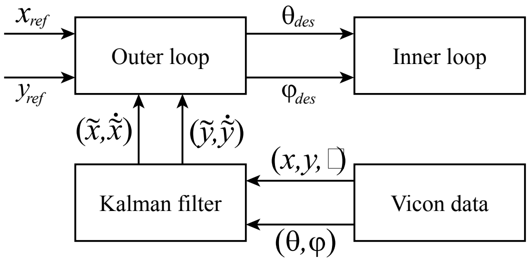

Fig. 13. General control schema

VI. Graphical user interface

Now that we already solved the two major problems (Path planning and feedback control), we want to offer a simple way to manipulate drone system. Consequently, any user can easily get benefit from the application that we have developed. A very simple way to do that is using Graphical User Interface Editor (GUIDE) in Matlab. The GUI must provide the following options

- Manual takeoff and Land buttons
- Emergency land
- Extra information including battery level and the drone state
- Real time localization of the system in the map
- Input the goal position and generate the 2D path
- Follow the generated path

The graphical user interface design is based on callback functions to execute desired tasks. Callback functions are executed once the user clicks on the corresponding button.

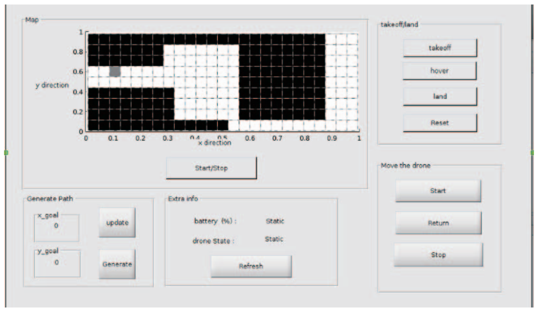

Fig. 14. Graphical user interface

Let's explain the button functions shown in Fig. 14. Basically, 5 major blocks are illustrated in the GUI: Map, Generate path, Extra info, Takeoff/Land, Move the drone. The Map block enables the user to input the goal position by clicking on one of the white squares of the map image and provides a real time localization of the drone. Generate path block uses the goal position as input and generate a possible path from the drone initial. Takeoff/Land block affords manual takeoff and land commands, it also offers an emergency button 'reset' that stops the drone motors during the flight and reset the software running onboard, and a 'hover' button in case when the user wants to keep the drone in fixed position. Extra info block shows information about battery level and drone state. The drone battery level is of key importance for insuring safe flights and avoiding accidental power off.

VII. Experiments

We conducted a service experiment in B-sen. The service scenario of the experiment is as follows. Firstly, someone stands in front of the entrance door and tries to enter into the room. Then the user requests to ROS-TMS by a GUI device to send a drone to the entrance so that the user can see what is happening . The captured video by the drone is displayed on a screen in a room. Figures 15 and 16 show the map and the planned trajectory by the proposed system. Figure 17 shows the experimental results. Left images show the motion of a drone and right images show the GUI interface. After suggesting the destination by pointing on the GUI, the drone took off, moved along the desired path, and reached to the destination automatically.

VIII. Conclusions

In this paper, we have been solving path planning and feedback control problems in an Informationally Structured Environment based on ROS-TMS framework. The major problems solved are a path planning using Dijkstra's algorithm and occupancy grid map, trajectory tracking problem using dynamic model of the drone and some control theory results. We conducted some service experiments by a drone using a developed GUI interface. Some assumptions have been done to design the path planning algorithm can cause some problems in specific cases. For instance, the assumption

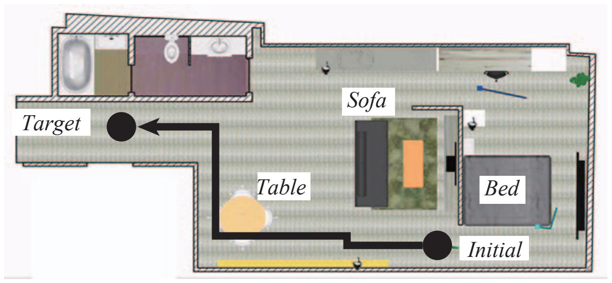

Fig. 15. Room map and initial and target positions

of a static environment is not valid if obstacle positions are modified or new unexpected obstacles are added to the structure. Hence, a dynamic map of the environment, updated each time the algorithm is executed, is of particular value and solves this problem.

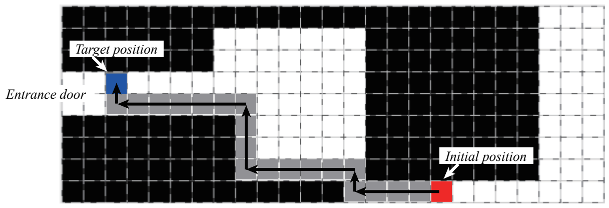

Fig. 16. Planned trajectory by the developed system

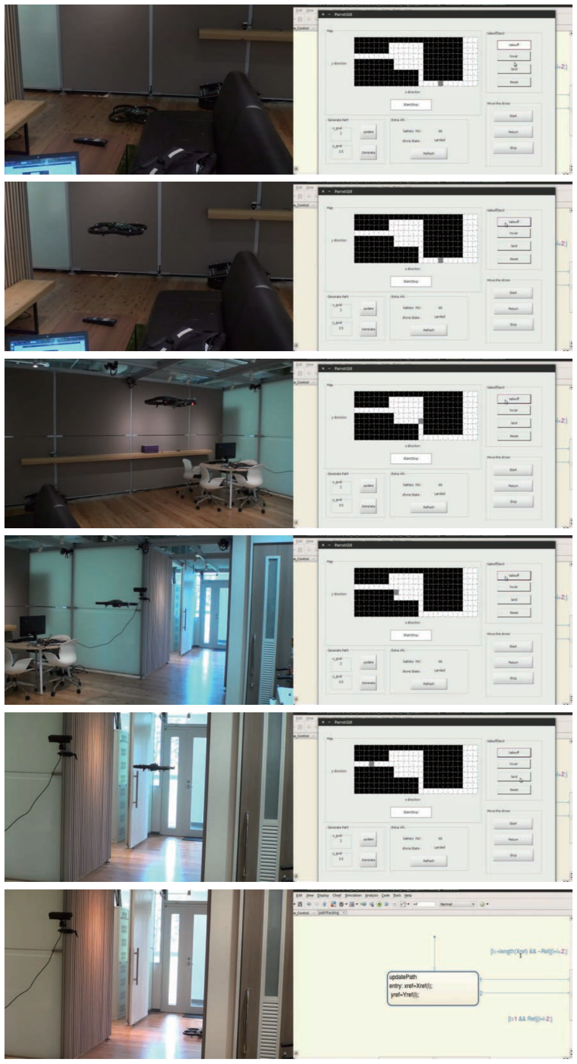

Fig. 17. Experiments

ACKNOWLEDGMENTS

This research is supported by The Japan Science and Technology Agency (JST) through its "Center of Innovation Science and Technology based Radical Innovation and Entrepreneurship Program (COI Program)."

REFERENCES

[1] Anuj Puri. A survey of unmanned aerial vehicles (uav) for traffic surveillance. *Department of computer science and engineering, University of South Florida*, 2005.

[2] Vladimir Reilly, Haroon Idrees, and Mubarak Shah. Detection and tracking of large number of targets in wide area surveillance. In *Computer Vision–ECCV 2010*, pages 186–199. Springer, 2010.

[3] Cristina Gomez and David R. Green. Small-scale airborne platforms for oil and gas pipeline monitoring and mapping. *University of Aberdeen report*, 2015.

[4] Y. Iwashita, A. Stoica, and R. Kurazume. Finding people by their shadows: Aerial surveillance using body biometrics extracted from ground video. In *Emerging Security Technologies (EST), 2012 Third International Conference on*, pages 43–48, Sept 2012.

[5] Parth N Patel, Malav A Patel, Rahul M Faldu, and Yash R Dave. Quadcopter for agricultural surveillance. *Advance in Electronic and Electric Engineering*, 3(4):427–432, 2013.

[6] J. Berni, P.J. Zarco-Tejada, L. Suarez, and E. Fereres. Thermal and narrowband multispectral remote sensing for vegetation monitoring from an unmanned aerial vehicle. *Geoscience and Remote Sensing, IEEE Transactions on*, 47(3):722–738, March 2009.

[7] Yoonseok Pyo, Kouhei Nakashima, Shunya Kuwahata, Ryo Kurazume, Tokuo Tsuji, Ken'ichi Morooka, and Tsutomu Hasegawa. Service robot system with an informationally structured environment. *Robotics and Autonomous Systems*, 2015.

[8] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The grasp multiple micro-uav testbed. *Robotics & Automation Magazine, IEEE*, 17(3):56–65, 2010.

[9] Lionel Heng, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 2472–2477. IEEE, 2011.

[10] Daniel Gurdan, Jan Stumpf, Michael Achtelik, Klaus-Michael Doth, Gerd Hirzinger, and Daniela Rus. Energy-efficient autonomous four-rotor flying robot controlled at 1 khz. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 361–366. IEEE, 2007.

[11] Tokuo Tsuji, Oscar Martinez Mozos, Hyunuk Chae, Yoonseok Pyo, Kazuya Kusaka, Tsutomu Hasegawa, Ken'ichi Morooka, and Ryo Kurazume. An informationally structured room for robotic assistance. *Sensors*, pages 1–28, 2015.

[12] ardrone_autonomy. http://ardrone-autonomy.readthedocs.org/en/latest/.

[13] Parrot. http://ardrone2.parrot.com/ardrone-2/specifications/.

[14] Ronald Arkin. *Behavior-Based Robotics*. The MIT Press, 1998.

[15] Engel Jakob, Sturm Jürgen, and Cremers Daniel. Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems (RAS)*, pages 1–14, 2014.