ROS-Based **M**ulti-**A**gent **S**ystems **CO**ntrol Hybrid **T**estbed (MASCOT)

Arvind Pandit and Ameer K. Mulla

***Abstract*— This paper presents a hybrid multiagent system simulation testbed with human in the loop control which is developed for testing and demonstration of different linear and non-linear decentralized control algorithms designed for multiagent system.This paper is aimed at the development of the platform which bridges a gap between theory and practical for the deployment of such algorithms. This platform supports Human in loop control where the leader are bilaterally teleoperated using a 3-DOF Novint Falcon haptic controller. The system developed provides the testbed for the simulation of multiagent systems having quadcopter agents. Here the quadcopter are assummed to have a simple dynamics as a simple and double integrator system. This is based on the fact that under certain assumptions, quadcopter dynamics can be modeled as a double integrator system. There is also hybrid simulation structure in the platform which enables user to simulate agents a mixture of virtual and real agents. Crazyflie 2.1 is used a hardware agent for the hybrid simulation. A gazebo simulator with a physics engine as ODE (Open Dynamics Engine) is used for simulating the dynamics of the quadcopter. Robot Operating System is used to develop communication networks and motion control algorithms for the different agents in the simulation testbed. The performance of the test bed is analyzed by implementing linear control laws such as position control, leaderless consensus, leader-follower consensus, bearing based formation and non-linear control law for min-max time consensus. This work is published as an open-source ROS package under MIT license at `https://github.com/Avi241/mascot`. A docker image is also developed for easy setup of the system.**

I. Introduction

II. Preliminaries

In this section, first, we describe the preliminaries of the dynamics of a quadcopter and review how a quadcopter can be represented using double integrator dynamics. A few preliminaries used in distributed control of multi-agent systems are also discussed.

A. Frame of References

Figure 1 shows the standard definitions [1] of the reference frames used in this simulation. $\{E\}$ is the *Earth reference frame* having its z-axis facing upward in the space. The body coordinate frame, $\{B\}$ is attached to the center of the quadcopter having its z-axis facing downwards in the space. $\{B'\}$ reference frame is attached to the body having its origin same as $\{B\}$ and $x^{B'}y^{B'}$ plane is parallel to $x^E y^E$.

The angles θ_p (pitch), θ_r (roll), and θ_y (yaw) represents the rotation of the body along x, y, and z-axis of the reference frame respectively. As $x^{B'}y^{B'}$ plane is parallel to $x^E y^E$ thus $\{B'\}$ can only rotate along the z-axis with angle θ_p. The

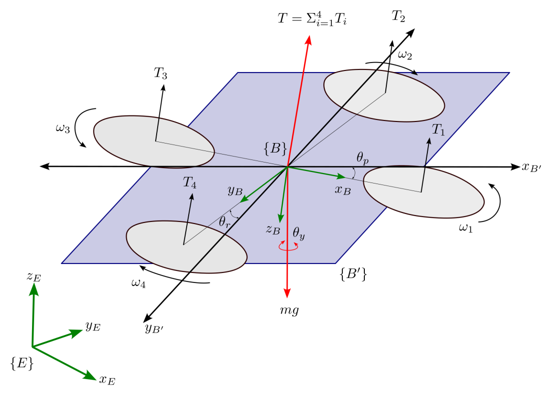

Fig. 1. Quadcopter reference frames setup

change in orientation between $\{B'\}$ and $\{E\}$ is given by a rotation matrix

$$\mathbf{R}^E_{B'} = \begin{bmatrix} c\theta_y & s\theta_y & 0 \\ -s\theta_y & c\theta_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $c(.) := \cos(.)$ and $s(.) := \sin(.)$

B. Quadcopter Dynamics

The upward thrust along the $-z^B$ axis is given by

$$T_i = b\bar{\omega}_i^2$$

where ω_i is the angular velocity of the rotor and $b > 0$ is a lift constant that depends on the air density, number of blades, cube of rotor blade radius, and chord length of the blade. According to Newton's second law of motion, the translation dynamics of the quadcopter in $\{E\}$ is given by

$$m\dot{\mathbf{v}}^E = \begin{bmatrix}0 & 0 & mg\end{bmatrix}^T - \mathbf{R}^E_B \begin{bmatrix}0 & 0 & T\end{bmatrix}^T - B\mathbf{v} \qquad (1)$$

where $\mathbf{v}$ is the velocity of quadcopter in {E} given by $\mathbf{v} = \begin{bmatrix}{v_x}^E & {v_y}^E & {v_z}^E\end{bmatrix}^T \in \mathbb{R}^3$, B is aerodynamics friction and $T = \Sigma T_i$ is total upward thrust generated by the rotors in $-z^B$ direction, g is the gravitational acceleration, and m is the mass of quadcopter. In (1), first term represents the gravitational force acting in $-z^E$ direction, second term represents the total upward thrust generated in $-z^B$ direction in $\{B\}$ and rotated in $\{E\}$.

The rotation in each axis is obtained by varying pairwise differences in rotor thrust. Torque about x and y axes is given by

$$\tau_x = dT_4 - dT_2 = db(\bar{\omega}_4^2 - \bar{\omega}_2^2)$$

$$\tau_y = dT_1 - dT_3 = db(\bar{\omega}_1^2 - \bar{\omega}_3^2)$$

The torque applied to each motor is opposed by aerodynamic drag and exerts a reaction torque on the frame which is given

Arvind Pandit and Ameer K. Mulla are with the Department of Electrical Engineering, Indian Institute of Technology Dharwad, Karnataka, India. Email: `arvind.pandit.21@iitdh.ac.in`, `ameer@iitdh.ac.in`

by $Q_i = k\bar{\omega}_i$ where k depends on the same factors as b. The total reaction torque about z-axis is given by

$$\tau_z = Q_1 - Q_2 + Q_3 - Q_4 = k\left(\bar{\omega}_1^2 + \bar{\omega}_3^2 - \bar{\omega}_2^2 - \bar{\omega}_4^2\right) \quad (2)$$

The total torque applied to the quadcopter body is $\mathbf{\Gamma} = \begin{bmatrix}\tau_x & \tau_y & \tau_z\end{bmatrix}$. By Euler's equation of motion rotational acceleration is given is by

$$J\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \mathbf{\Gamma} \quad (3)$$

where J is the 3×3 inertia matrix of the quadcopter and ω is a angular velocity vector of the quadcopter body frame. Overall quadcopter motion equation [1] is obtained by integrating equations (1) and (2) which is given by

$$\begin{bmatrix}\mathbf{T} & \mathbf{\Gamma}\end{bmatrix}^T = A\begin{bmatrix}\bar{\omega}_1^2 & \bar{\omega}_2^2 & \bar{\omega}_3^2 & \bar{\omega}_4^2\end{bmatrix}^T \quad (4)$$

where $A = \begin{bmatrix}-b & -b & -b & -b\\ 0 & -db & 0 & db\\ db & 0 & -db & 0\\ k & -k & k & -k\end{bmatrix}$ is a constant invertible matrix since $b, k, d > 0$.

C. Quadcopter Dynamics as Double Integrators

The overall motion in the plane $x^{B'}y^{B'}$ is obtained by pitching and rolling the quadcopter by an angle θ_p and θ_r. The total force acting on a quadcopter is given by

$$\mathbf{f}^{B'} = \mathbf{R}x\left(\theta r\right)\mathbf{R}y\left(\theta p\right)\begin{bmatrix}0 & 0 & T\end{bmatrix}^T \quad (5)$$

where,

$$\mathbf{R}x\left(\theta r\right) = \begin{bmatrix}1 & 0 & 0\\ 0 & c\theta_r & s\theta_r\\ 0 & -s\theta_r & c\theta_r\end{bmatrix},\ \mathbf{R}y\left(\theta p\right) = \begin{bmatrix}c\theta_p & 0 & s\theta_p\\ 0 & 1 & 0\\ -s\theta_p & 0 & c\theta_p\end{bmatrix}$$

are the Rotation matrix about x and y axes respectively. Simplifying (5) we get $\mathbf{f}^{B'}$ as

$$\mathbf{f}^{B'} = \begin{bmatrix}T\sin\theta_p & T\sin\theta_r\cos\theta_p & T\cos\theta_r\cos\theta_p\end{bmatrix}^T$$

for small θ_p and θ_r the above equation can be approximated by

$$\mathbf{f}^{B'} \approx \begin{bmatrix}T\theta_p & T\theta_r & T\end{bmatrix}^T$$

i.e $f_x^{B'} \approx T\theta_p$ and $f_y^{B'} \approx T\theta_r$
According to Newton's Second law of motion $F = ma$, we can write

$$ma_x^{B'} = T\theta_p$$

$$\theta_p = \frac{m}{T}a_x^{B'},\quad \theta_r = \frac{m}{T}a_y^{B'}$$

where $a_x^{B'}$ is $a_y^{B'}$ is acceleration of quadcopter in $\{B'\}$ frame.
As the controller will receive the position and velocity in $\{E\}$ frame so it will compute the acceleration in $\{E\}$ frame which needs to be rotated to the $\{B'\}$ frame.

$$\mathbf{f}^{B'} = \mathbf{R}_E^{B'}\mathbf{f}^{\mathbf{E}}$$

$$m\mathbf{a}^{B'} = \mathbf{R}_E^{B'}m\mathbf{a}^{\mathbf{E}}$$

$$\mathbf{a}^{B'} = \mathbf{R}_E^{B'}\mathbf{a}^{\mathbf{E}}$$

This equation shows the motion of the quadcopter in $x^{B'}y^{B'}$ plane.

Remark 1: While the discussion above describes the motion of the quadcopter in $x^{B'}y^{B'}$ plane as a double integrator, motion along $-z^{B'}$ is directly dependent on the thrust generated by the rotors in {B} frame given by $T = \Sigma T_i$, which is naturally a double integrator.

D. Preliminaries of Distributed Control

A homogeneous multi-agent system is a collection of n agents with identical dynamics communicating with each other over a communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the set of vertices $\mathcal{V} = \{\alpha_i,\ i = 1, ..., n\}$ represents the agents and the edges $(\alpha_i, \alpha_j) \in \mathcal{E}$ denote the communication from agent α_i to α_j. The communication graph $\mathcal{G}$ is *undirected* if $(\alpha_i, \alpha_j) \in \mathcal{E} \implies (\alpha_j, \alpha_i) \in \mathcal{E}$ otherwise it is *directed.* A *path* in a graph is a sequence of edges. An undirected graph is said to be *connected* if it has a path from each agent to every other agent. A directed graph is said to contain a *directed spanning tree* if there is an agent α_r (called *root*) such that there is a directed path from α_r to every other agent. Set of neighbours for each agent is given by $\mathcal{N}_i := \{j \in \mathcal{V} : (\alpha_i, \alpha_j) \in \mathcal{E}\}$. The Laplacian matrix $\mathcal{L}$ of a graph $\mathcal{G}$ is given by $\mathcal{L}_n = [lij] \in \mathbb{R}^{n\times n}; l_{ij} = -a_{ij}, i \neq j, l_{ii} = \sum_{j=1,j\neq i}^{n} a_{ij}$, where a_{ij} is the *weight* of the edge (α_i, α_j). Some control algorithms use leader-follower configuration of the multi-agent systems in which, the set of agents is divided into a set of leader $\mathbf{L}$ and a set of followers $\mathbf{F}$.

In this paper, agents are assumed to have double-integrator dynamics, given by

$$\ddot{\mathbf{x}}_i^E(t) = \mathbf{f}_i^E(t) \quad (6)$$

where $\mathbf{x}_i^E = \begin{bmatrix}\mathbf{p}_i^E\\ \mathbf{v}_i^E\end{bmatrix} \in \mathbb{R}^{2d}$ is the state vector of α_i and $\mathbf{f}_i^E \in \mathbb{R}^d$ denotes the input to α_i. The vectors $\mathbf{p}_i^E$ and $\mathbf{v}_i^E$ denote the position and velocity of α_i, respectively, in $\{E\}$ frame.

$$f_i^E = \left\{\begin{array}{l}\sum_{j=1}^{n}\\ 0 \text{ if } a_i \in \mathcal{A}\end{array}\right. \quad (7)$$

where a_{ij} is the (i,j) entry of the adjacency matrix $A_n \in \mathbb{R}^{n\times n}$ corresponding to the communication graph $\mathcal{G}_n$ and β is a positive constant.This control law is the modified version of the control law mentioned in [2].The derivation of this law is provided in [3].The leader position is seprately controlled by a waypoint Navigation Controller.The control law discussed here drives **n** quadcopters to consensus i.e for any initial p_i^E and v_i^E and for all i,j = 1,....,n i.e $||p_i^E(t) - p_j^E(t)|| \to 0$ and $v_i^E(t) \to 0$ as $t \to \infty$.

Remark 2: The heading control loop given in the next section is designed such that $\theta_y \to \theta_y^*$ relatively fast and is held constant. Thus, matrix $\mathbf{R}_{B'}^E$ can be considered as constant.

E. Preliminaries of Bearing based Formation Control

F. Preliminaries of Human-in-the-loop Control

Human-in-the-loop (HITL) refers to a type of system in which a human operator is involved in the control loop of a robotic system. In a robotics simulator, human-in-the-loop typically means that a human operator can interact with the simulated robotic system in real time and make decisions or provide input that affects the behavior of the simulated robot. There are many potential applications for human-in-the-loop robotics simulations. For example, a robotics simulator with human-in-the-loop functionality could be used to test and evaluate the performance of a new robotic system, or to train operators to work with the system in a safe and controlled environment. It could also be used to test and validate control algorithms or other software that will be used to operate the real-world robotic system.

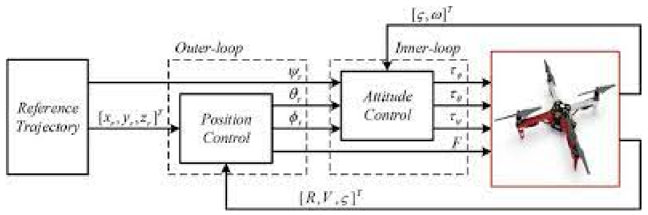

Fig. 2. Human-in-the-loop control Block diagram

III. MASCOT: Structure and Features

A. Tools Used

1) Robot Operating System (ROS): ROS is an open-source framework that helps researchers and developers build and reuse code between robotics applications. ROS provides a distributive architecture that eliminates the problem of a single node handling all the tasks. ROS supports low-level drivers and programming in various languages such as Python, C++, Java, and Lisp. This makes it convenient to directly deploy the codes written for simulation on the physical systems. Visualization and debugging tools such as Rviz and rqt make it very easy to keep track of the process [4]. The basic architecture of ROS is shown in Fig. 3.

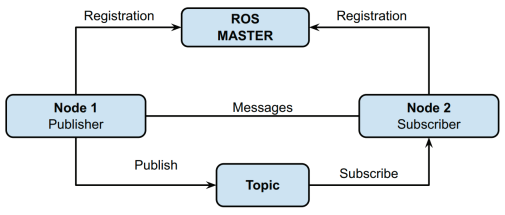

Fig. 3. ROS Architecture

2) Gazebo: The gazebo is a 3D simulation platform that provides robots, sensors, and environment models which are required for robot development. It uses Open Dynamics Engine (ODE) for real-time simulations and graphics. Gazebo supports a wide range of sensors such as LRF,2D/3D camera, Depth Camera, and Force-Torque Sensor. APIs provided in Gazebo enable users to create robots, sensors, and environment control as a plug-in.

3) Falcon Haptic Controller: We are using Novint Falcon as an input for Human-in-the-loop simulation.The Novint Falcon is a haptic device that is designed to provide users with the ability to touch and feel virtual objects in a computer-generated environment. It is typically used in gaming, virtual reality, and simulation applications.The Novint Falcon uses a series of motors and linkages to create force feedback and motion, allowing users to experience the sensation of touch and motion in virtual environments. It has three degrees of freedom (DOF), allowing it to move in the x, y, and z axes. It also has a number of buttons and controls that can be used to interact with the virtual environment.

One of the unique features of the Novint Falcon is its ability to create a wide range of haptic sensations, including force feedback, vibrations, and thermal effects. This allows it to provide a more immersive and realistic experience for users, making it useful for a variety of applications, including gaming, training, and simulation. Overall, the Novint Falcon is a powerful haptic device that allows users to touch and feel virtual objects in a computer-generated environment, providing a more immersive and realistic experience. It is useful for a variety of applications, including gaming, training, and simulation, and its ability to create a wide range of haptic sensations makes it particularly useful for creating more realistic and engaging virtual environments.

Fig. 4. Falcon Haptic Controller

4) Crazyflie Quadcopter: The harware quadcopter used in the testbed for hybrid simulation is Crazyflie 2.1. The Crazyflie 2.1 is a small and lightweight drone developed by Bitcraze, a Swedish robotics company. The Crazyflie 2.1 is equipped with a number of sensors and features that allow it to navigate and stabilize itself in flight. It has an onboard camera, gyroscopes, accelerometers, and magnetometers, as well as a barometer and ultrasound rangefinder. It also has a wireless communication system, allowing it to be controlled remotely using a radio controller or a computer.The Crazyflie 2.1 quadcopter measures 92 millimeters between diagonally opposed motor shafts and weighs 27 grams with a battery. It contains a 32-bit, 168- MHz ARM microcontroller with floating-point unit that is capable of significant onboard computation. Software and hardware are both open-source. The Crazyflie communicates with a PC over the Crazyradio PA, a 2.4 GHz USB radio that transmits up to two megabits per second in 32-byte packets. The Crazyflie 2.1 is capable of

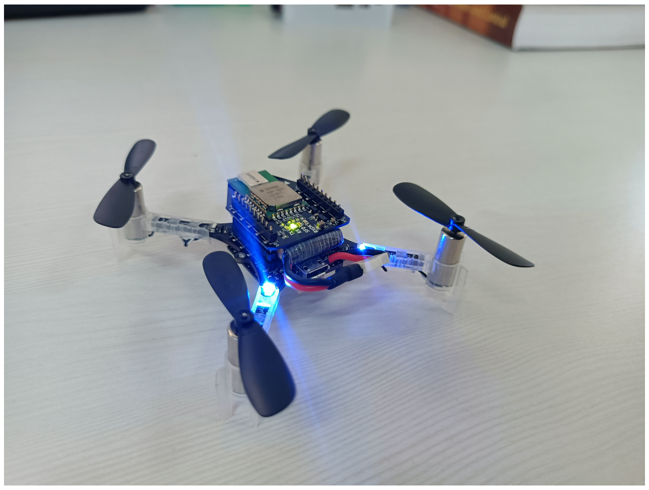
Fig. 5. Crazyflie Quadcopter

flying for up to 7 minutes on a single battery charge, and can reach speeds of up to 10 meters per second. It is small and lightweight, weighing just 27 grams, and is capable of flying indoors and outdoors. We have added Optitrack markers and flow deck which increases its weight to 33 grams and reduces the flight time to 6 minutes. [5]

5) Optitrack Localization System:

6) Quadcopter Simulation Package: The quadcopter model used in MASCOT is based on the tum-simulator AR Parrot Drone Gazebo simulation package [6] developed by the Computer Vision Group at the Technical University of Munich. The lower level controller is modified and tuned so that the quadcopters can be treated as a double integrator system for control algorithm implementation, and added topics and controls required for the implemented model. A general control script is written for the implementation of the different use cases discussed in Section IV;

B. Control Block

1) Heading Control:

2) Position Control:

C. Architecture and Features of MASCOT

D. Configuring the Simulation

E. Configuring the Hybrid - Simulation

IV. Examples

A. Simulation with HITL

1) Waypoint Navigation:

2) Consensus Algorithms:

3) Min-max time Consensus Control:

4) Formation Control:

B. Hybrid Simulation with HITL

1) Waypoint Navigation:

2) Consensus Algorithms:

3) Min-max time Consensus Control:

4) Formation Control:

V. Conclusion and Future Work

References

[1] P. I. Corke and O. Khatib, *Robotics, vision and control: fundamental algorithms in MATLAB*. Springer, 2011, vol. 73.

[2] W. Ren and R. W. Beard, *Distributed consensus in multi-vehicle cooperative control*. Springer, 2008, vol. 27.

[3] A. Joshi, N. Limbu, I. Ahuja, A. K. Mulla, H. Chung, and D. Chakraborty, "Implementation of distributed consensus on an outdoor testbed," in *2016 European Control Conference (ECC)*, 2016, pp. 2146–2151.

[4] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[5] W. Hönig and N. Ayanian, *Flying Multiple UAVs Using ROS*. Cham: Springer International Publishing, 2017, pp. 83–118. [Online]. Available: https://doi.org/10.1007/978-3-319-54927-9_3

[6] T. C. V. Group, "Tum simulator," https://github.com/tum-vision/tum_simulator, 2012.