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

Arvind Pandit, Akash Njattuvetty, and Ameer K. Mulla

Department of Electrical Engineering
Indian Institute of Technology Dharwad, India

Indian Control Conference (ICC-8)
14-16 December 2022, Chennai, India.

# Overview

# Multi-Agent Systems

# Multi-Agent Systems

**Multi-Agent Systems:**

A system consists of multiple co-operative agents interacting with each other.

**Distributed Control:**

- Control is distributed among multiple agents.
- Each agent with it's own local control algorithm.
- Communicating with each other.

# Multi-Agent Systems

**Multi-Agent Systems:**

A system consists of multiple co-operative agents interacting with each other.

**Distributed Control:**

- Control is distributed among multiple agents.
- Each agent with it's own local control algorithm.
- Communicating with each other.

**Advantages:**

- Achieve complex objectives.
- Faster exploration.

# Multi-Agent Systems

**Multi-Agent Systems:**

A system consists of multiple co-operative agents interacting with each other.

**Distributed Control:**

- Control is distributed among multiple agents.
- Each agent with it's own local control algorithm.
- Communicating with each other.

**Advantages:**

- Achieve complex objectives.
- Faster exploration.

**Applications:**

Robotics, space missions, search and exploration, surveillance, agriculture etc.

# Simulation Platform for Multiagent System

**Simulation:**

- Test the performance of robot before it is built.
- Evaluate different control laws.
- Train in safe and controlled environment.
- Study the behaviour of the system.

# Simulation Platform for Multiagent System

**Simulation:**

- Test the performance of robot before it is built.
- Evaluate different control laws.
- Train in safe and controlled environment.
- Study the behaviour of the system.

**Existing MAS Simulation:**

- MATLAB based simulators.
- Limitation of no. of agents.
- Not readily deployable of hardware.

# MASCOT

- Developed using open source tools.
- ROS and Gazebo.
- Supports low level driver.
- Simple user interface.
- In this version Quadcopter as an agent.
- Multiagent system with double integrator.
- Easy to setup with Docker Support.

# MASCOT

- Developed using open source tools.
- ROS and Gazebo.
- Supports low level driver.
- Simple user interface.
- In this version Quadcopter as an agent.
- Multiagent system with double integrator.
- Easy to setup with Docker Support.

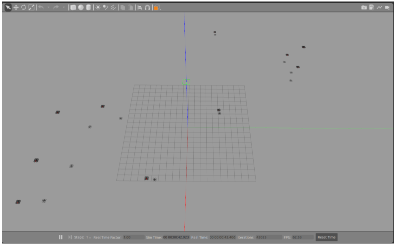

Figure: Initial Position of Drones

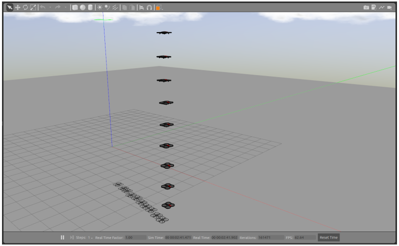

Figure: Final Position

# Preliminaries

# Frame of Reference[1]

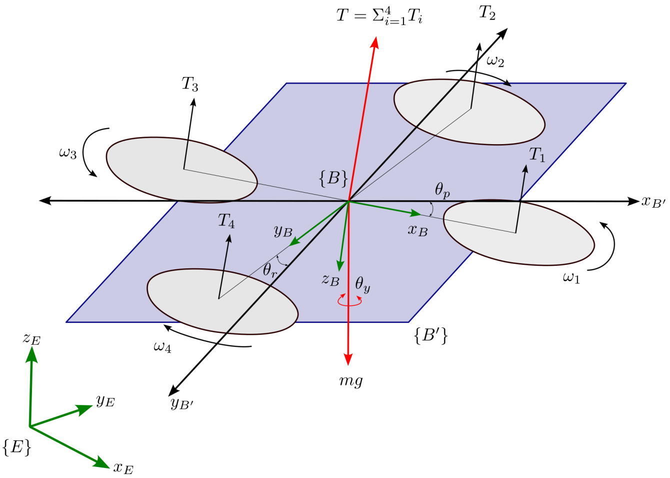


Figure: Frame of Reference

[1] P. I. Corke *et al.*,Robotics, vision and control: fundamental algorithms in MATLAB

# Quadcopter Dynamics

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

$$\mathbf{T} = \Sigma(b\omega_i^2) \tag{1}$$

# Quadcopter Dynamics

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

$$\mathbf{T} = \Sigma(b\omega_i^2) \tag{1}$$

- The translation dynamics of the quadcopter in $\{E\}$ is

$$m\dot{\mathbf{v}}^E = \begin{bmatrix}0 & 0 & mg\end{bmatrix}^T - \mathbf{R}_B^E \begin{bmatrix}0 & 0 & T\end{bmatrix}^T - B\mathbf{v} \tag{2}$$

# Quadcopter Dynamics

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

$$\mathbf{T} = \Sigma(b\omega_i^2) \tag{1}$$

- The translation dynamics of the quadcopter in $\{E\}$ is

$$m\dot{\mathbf{v}}^E = \begin{bmatrix}0 & 0 & mg\end{bmatrix}^T - \mathbf{R}_B^E \begin{bmatrix}0 & 0 & T\end{bmatrix}^T - B\mathbf{v} \tag{2}$$

- Torque about $x$ and $y$ axes is

$$\tau_x = dT_4 - dT_2 = db(\bar{\omega}_4^2 - \bar{\omega}_2^2) \tag{3}$$

$$\tau_y = dT_1 - dT_3 = db(\bar{\omega}_1^2 - \bar{\omega}_3^2) \tag{4}$$

# Quadcopter Dynamics

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

$$\mathbf{T} = \Sigma(b\omega_i^2) \tag{1}$$

- The translation dynamics of the quadcopter in $\{E\}$ is

$$m\dot{\mathbf{v}}^E = \begin{bmatrix}0 & 0 & mg\end{bmatrix}^T - \mathbf{R}_B^E \begin{bmatrix}0 & 0 & T\end{bmatrix}^T - B\mathbf{v} \tag{2}$$

- Torque about $x$ and $y$ axes is

$$\tau_x = dT_4 - dT_2 = db(\bar{\omega}_4^2 - \bar{\omega}_2^2) \tag{3}$$

$$\tau_y = dT_1 - dT_3 = db(\bar{\omega}_1^2 - \bar{\omega}_3^2) \tag{4}$$

- The total reaction torque about $z$-axis is

$$\tau_z = Q_1 - Q_2 + Q_3 - Q_4 = k\left(\bar{\omega}_1^2 + \bar{\omega}_3^2 - \bar{\omega}_2^2 - \bar{\omega}_4^2\right) \tag{5}$$

# Quadcopter Dynamics

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

$$\mathbf{T} = \Sigma(b\omega_i^2) \tag{1}$$

- The translation dynamics of the quadcopter in $\{E\}$ is

$$m\dot{\mathbf{v}}^E = \begin{bmatrix}0 & 0 & mg\end{bmatrix}^T - \mathbf{R}_B^E \begin{bmatrix}0 & 0 & T\end{bmatrix}^T - B\mathbf{v} \tag{2}$$

- Torque about $x$ and $y$ axes is

$$\tau_x = dT_4 - dT_2 = db(\bar{\omega}_4^2 - \bar{\omega}_2^2) \tag{3}$$

$$\tau_y = dT_1 - dT_3 = db(\bar{\omega}_1^2 - \bar{\omega}_3^2) \tag{4}$$

- The total reaction torque about $z$-axis is

$$\tau_z = Q_1 - Q_2 + Q_3 - Q_4 = k\left(\bar{\omega}_1^2 + \bar{\omega}_3^2 - \bar{\omega}_2^2 - \bar{\omega}_4^2\right) \tag{5}$$

- By Euler's equation of motion rotational acceleration is

$$J\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{\Gamma} \tag{6}$$

# Quadcopter Dynamics

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

$$\mathbf{T} = \Sigma(b\omega_i^2) \tag{1}$$

- The translation dynamics of the quadcopter in $\{E\}$ is

$$m\dot{\mathbf{v}}^E = \begin{bmatrix} 0 & 0 & mg \end{bmatrix}^T - \mathbf{R}_B^E \begin{bmatrix} 0 & 0 & T \end{bmatrix}^T - B\mathbf{v} \tag{2}$$

- Torque about $x$ and $y$ axes is

$$\tau_x = dT_4 - dT_2 = db(\bar{\omega}_4^2 - \bar{\omega}_2^2) \tag{3}$$

$$\tau_y = dT_1 - dT_3 = db(\bar{\omega}_1^2 - \bar{\omega}_3^2) \tag{4}$$

- The total reaction torque about $z$-axis is

$$\tau_z = Q_1 - Q_2 + Q_3 - Q_4 = k\left(\bar{\omega}_1^2 + \bar{\omega}_3^2 - \bar{\omega}_2^2 - \bar{\omega}_4^2\right) \tag{5}$$

- By Euler's equation of motion rotational acceleration is

$$J\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{\Gamma} \tag{6}$$

- Overall quadcopter motion equation

$$\begin{bmatrix} \mathbf{T} & \boldsymbol{\Gamma} \end{bmatrix}^T = A\begin{bmatrix} \bar{\omega}_1^2 & \bar{\omega}_2^2 & \bar{\omega}_3^2 & \bar{\omega}_4^2 \end{bmatrix}^T \tag{7}$$

# Quadcopter Dynamics as Double Integrator

- Total force on quadcopter

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

- Thus we get $\mathbf{f}^{B'}$ as

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

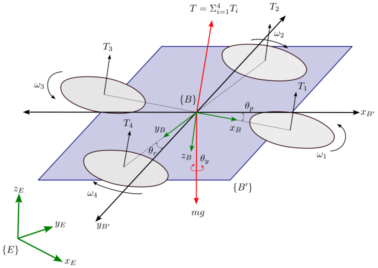


Figure: Quadcopter Dynamics

# Quadcopter Dynamics as Double Integrator

- Total force on quadcopter

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

- Thus we get $\mathbf{f}^{B'}$ as

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

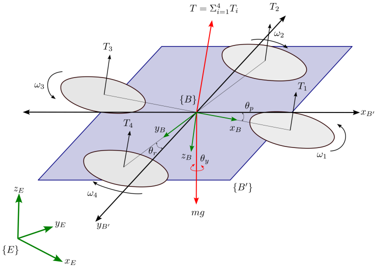


Figure: Quadcopter Dynamics

- For small $\theta_p$ and $\theta_r$ the $\mathbf{f}^{B'}$ can be approximated by

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

- With this assumption the Quadcopter can be assumed as a double integrator system where $\theta_p$ and $\theta_r$ are given by

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

# MASCOT:Structure and Features

# Tools Used

**ROS:**

- Open source robotics framework.
- Distributed architecture with intercommunication between different nodes.
- Support for various programming language Python, C++, Java.
- Rich visualization and debugging tools.

# Tools Used

**ROS:**

- Open source robotics framework.
- Distributed architecture with intercommunication between different nodes.
- Support for various programming language Python, C++, Java.
- Rich visualization and debugging tools.

**Gazebo:**

- 3D simulation platform.
- Uses Open Dynamics Engine for realtime physics simulation.
- Support for sensor and actuator plugin.

# Tools Used

### ROS:

- Open source robotics framework.
- Distributed architecture with intercommunication between different nodes.
- Support for various programming language Python, C++, Java.
- Rich visualization and debugging tools.

### Gazebo:

- 3D simulation platform.
- Uses Open Dynamics Engine for realtime physics simulation.
- Support for sensor and actuator plugin.

### TUM Simulator Package:

- Uses the AR Parrot drone model.
- Low level plugin is modified as per the Double integrator dynamics.
- Added the required topics and controls.

# Control Block

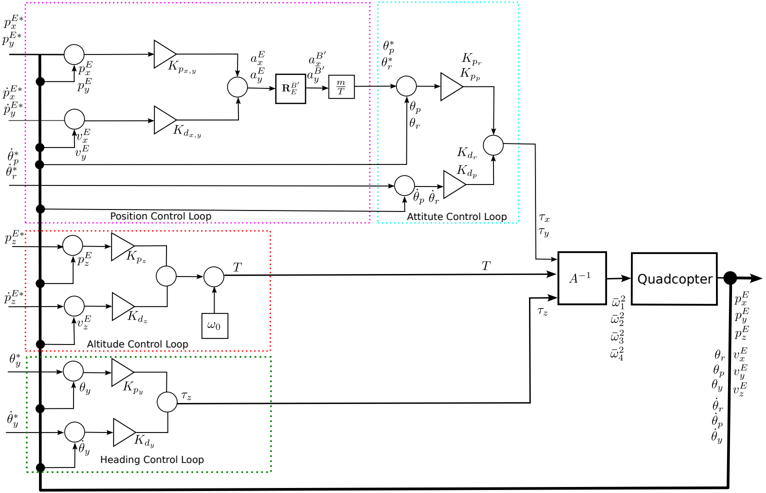


Figure: Control Block

# Architecture

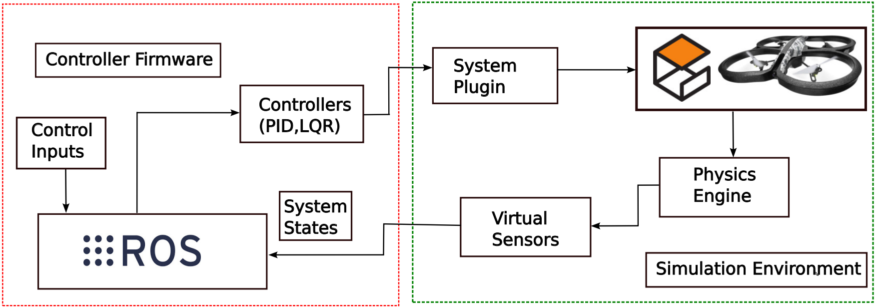


Figure: System Architecture

- Gazebo internal scheduler provides the ROS interface.
- ROS works as middleware which runs independent controller for each agent.
- The intercommunication uses TCPROS protocol.

# Feature and Configuration of Simulation Testbed

## Feature

- Easy modification.
- Supports multiple languages Python, Cpp, Java.
- Flexibility with no. of agents.

# Feature and Configuration of Simulation Testbed

## Feature

- Easy modification.
- Supports multiple languages Python, Cpp, Java.
- Flexibility with no. of agents.

## Configuration

- **Robot:** Details of the Robots to be simulated
  - **Number:** No. of Agents.
  - **IntialPosition:** Enable initializer.
  - **Position:** Initial Position.
- **Output:** Output config
  - **Velocity :** Generate Vel plot.
  - **Position :** Generate Vel plot.
  - **Save-plot :** Save plots.
  - **Show-plot :** Show plots.
  - **Save-data :** Save Numpy
- **Control:** Controls laws
  - **Custom-Control:**
  - **Tutorial Examples:**
  * **Waypoint Navigation:**
    - **P-Gain:** Default = 1.0
    - **D-Gain:** Default = 1.0
  * **Consensus:**
    - **Leader:** Robot index to be leader, 0-for leaderless.
    - **Communication Graph:**
    - **L-mat:** Laplacian Matrix.
  * **Min-max Consensus:**

# Examples

# Way-Point Navigation

- The position of the quadcopter in $x^{B'}y^{B'}$ plane is controlled independently by the proportional-derivative controller for each axis.

$$a_x = K_{p_x}\left(p_x^* - p_x\right) + K_{d_x}\left(\dot{p}_x^* - \dot{p}x\right)$$
$$a_y = K_{p_y}\left(p_y^* - p_y\right) + K_{d_y}\left(\dot{p}_y^* - \dot{p}y\right)$$

# Way-Point Navigation

- The position of the quadcopter in $x^{B'}y^{B'}$ plane is controlled independently by the proportional-derivative controller for each axis.

$$a_x = K_{p_x}\left(p_x^* - p_x\right) + K_{d_x}\left(\dot{p}_x^* - \dot{p}x\right)$$

$$a_y = K_{p_y}\left(p_y^* - p_y\right) + K_{d_y}\left(\dot{p}_y^* - \dot{p}y\right)$$

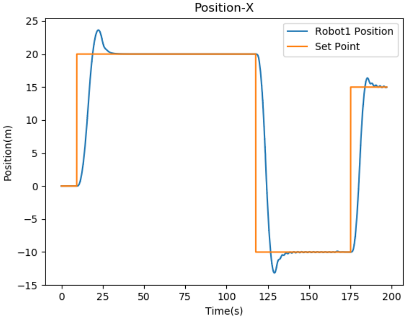


Figure: X-axis Position plot of Waypoint Navigation

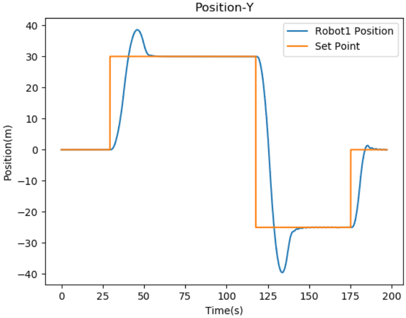


Figure: Y-axis Position plot of Waypoint Navigation

# Consensus Algorithm (Linear)

- A leaderless asymptotic consensus and leader follower is implemented.
- The control algorithms used is as follows:[2]

$$\mathbf{f}_i^E = \begin{cases} \sum_{j=1}^{n} a_{ij} \left( \mathbf{p} j^E - \mathbf{p} i^E \right) - \beta \mathbf{v}_i^E \text{ if } \alpha_i \in \mathbf{F} \\ \\ 0 \text{ if } \alpha_i \in \mathbf{L} \end{cases}$$

[2] A.Joshi *et al.* Implementation of distributed consensus on an outdoor testbed,2016 ECC

# Consensus Algorithm (Linear)

- A leaderless asymptotic consensus and leader follower is implemented.
- The control algorithms used is as follows:[2]

$$\mathbf{f}_i^E = \begin{cases} \sum_{j=1}^{n} a_{ij} \left(\mathbf{p}j^E - \mathbf{p}i^E\right) - \beta \mathbf{v}_i^E \text{ if } \alpha_i \in \mathbf{F} \\ 0 \text{ if } \alpha_i \in \mathbf{L} \end{cases}$$

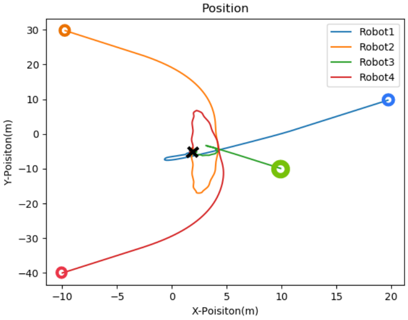


Figure: Leaderless Control plot

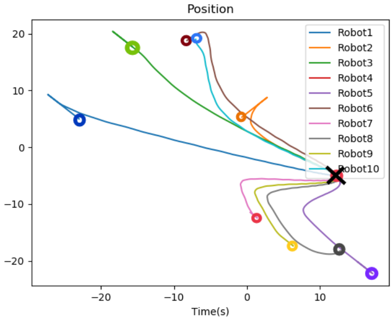


Figure: Leader Follower plot

[2] A.Joshi *et al.* Implementation of distributed consensus on an outdoor testbed,2016 ECC

# Min-Max Time Consensus

- A non linear Min-Max time consensus Algorithm is implemented.
- The Control Law used is [3]

$$\mathbf{f}_c^E = \beta_c \text{sign}(2(\beta_c - \beta_p)(\mathbf{p}_c - \mathbf{p}_p) + (\mathbf{v}_c - \mathbf{v}_p)^2 \text{sign}(\mathbf{v}_c - \mathbf{v}_p))$$

[3] A. Joshi *et al.* Implementation of min-max time consensus tracking on a multi-quadrotor testbed,2019 ECC

# Min-Max Time Consensus

- A non linear Min-Max time consensus Algorithm is implemented.
- The Control Law used is [3]

$$\mathbf{f}_c^E = \beta_c \text{sign}(2(\beta_c - \beta_p)(\mathbf{p}_c - \mathbf{p}_p) + (\mathbf{v}_c - \mathbf{v}_p)^2 \text{sign}(\mathbf{v}_c - \mathbf{v}_p))$$

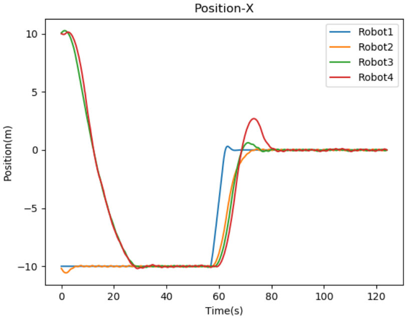


Figure: Position in X axis

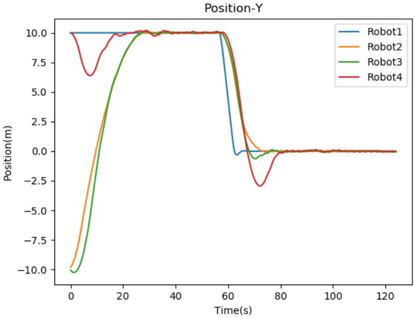


Figure: Position in Y axis

[3] A. Joshi *et al.* Implementation of min-max time consensus tracking on a multi-quadrotor testbed,2019 ECC

# Conclusion and Future Work

# Conclusion and Future Work

## Conclusion

- A simulation testbed for MultiAgent System.
- Quadcopter are approximated as a double integrator system.
- Various Linear and Non-Linear control laws are tested.
- This shows the integrity of the developed testbed.

# Conclusion and Future Work

## Conclusion

- A simulation testbed for MultiAgent System.
- Quadcopter are approximated as a double integrator system.
- Various Linear and Non-Linear control laws are tested.
- This shows the integrity of the developed testbed.

## Future Work

- System is Open-source and expandable.
- UGVs and different UAVs can be deployed.
- Human In the loop control.
- Deployment on real hardware.

# Thank You

Figure: https://github.com/Avi241/mascot