# Teleoperation of Multi-Agent Formations

Shengdong Liu

June 3, 2016

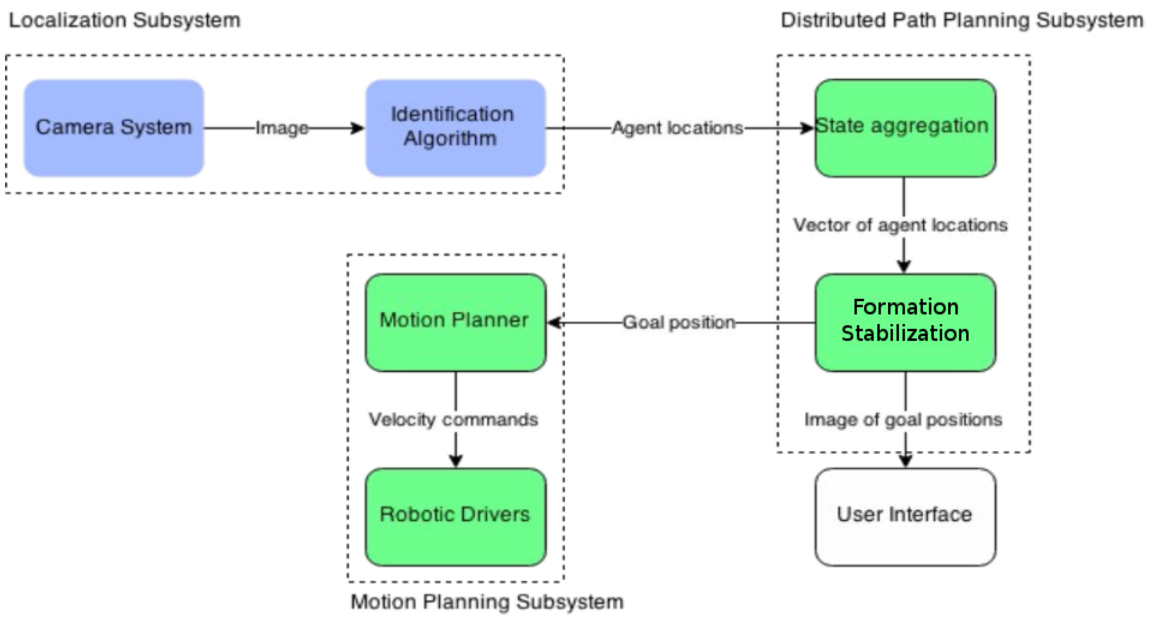
**Abstract**

As the Internet of things continues to grow and autonomous vehicles technologies continue to advance, a means to coordinate a network of autonomous vehicles is becoming increasingly important. This work aims to create a system that allows human interaction with a network autonomous agents by controlling the formation of the agents. We uses a global and robust formation stabilization algorithm[1], by Prof. Jorge Cortés at University of California San Diego, to stabilize the agents into a formation, and we used teleoperation commands to manipulate the formation itself. Both the formation stabilization algorithm and the teleoperation commands are implemented in Python, simulated in ROS (Robotic Operating System) framework and visualized in RVIZ (a 3D visualizer). The visual results shows that agents stabilize in the given formation which can be scaled and rotated via teleoperation.

## 1 Big Picture

As robotics technology advance, we are seeing an increasing the number of autonomous robots in our daily lives. This boom in robots calls for inter-robot communication and coordination to complete tasks in groups. To meet this need, the Multi-Agent Robotics Lab (MURO)[2] in UCSD, led by Professor Cortés and Martínez, has been incubating a testbed to test potential multi-agent control algorithms for deployment. The lab currently possess ten TurtleBots[3] and four Parrot AR drones[5] as a multi-agent robotic network. These equipments provide a stable system to which distributed control systems can be readily deployed for testing and validation on hardware.

We use the open source ROS (Robot Operating System)[4] as the software platform for its publisher-subscriber model to allow for inter-robot communication and coordination. The current system set up for the testbed consists of three subsystems. The localization subsystem provided the position and orientation of each of deployed agents using overhead cameras globally and SLAM algorithms locally. The distributed path planning system applies multi-agent control algorithms and uses the position of the agents to calculate their goal position. And Lastly, the motion planning subsystem navigates the agents to their respective goal positions. These subsystem are implemented as ROS nodes, and information are passed on between subsystems using ROS topics.



## 2 Personal Contribution

This quarter, I continued my work from the Wi2016 quarter, and work on a part of distributed path planning subsystem, which calculates the goal position of each agent based on their current positions. I implemented, in Python, an formation stabilization algorithm proposed by Prof. Cortés. In his paper "Global and robust formation-shape stabilization of relative sensing networks", Prof Cortés describes a simple distributed algorithm that achieves global stabilization of formations for relative sensing networks in arbitrary dimensions with fixed topology. Assuming the network runs an initialization procedure to equally orient all agent reference frames, convergence to the desired formation shape is guaranteed even in partially asynchronous settings[1]. After implementing the algorithm, I wrapped the code in a ROS node in order to sent the goal positions to the motion planner and receive teleoperation command from a user. I visualized the goal positions in RVIZ, a 3D visualizer for robotics, and the visualization shows that the agents stabilizes in the given formation which can be scaled and rotated via teleoperation commands.

## 3 Preliminaries

In this section, we introduce necessary notations to understand the formation stabilization algorithm and summarize the update rule used for the implementation.

For the details and proofs related to the algorithm, please refer to the paper "Global and robust formation-shape stabilization of relative sensing networks."[1]

### 3.1 Notations

let $n$ denotes the number of agents.

let $d$ denotes the dimension of Euclidean space.

let $G$ denotes a weighted digraph of all the agents.

let $D_{out}(G)$ denotes the $n \times n$ weighted out-degree matrix of the digraph $G$.

let $A(G)$ denotes the $n \times n$ weighted adjacency matrix with the following properties: for $i, j \in \{1, ..., n\}$, the entry $a_{ij} > 0$ if (i, j) is an edge of $G$, and $a_{ij} = 0$ otherwise.

let $L(G) = D_{out}(G) - A(G)$ denotes the graph Laplacian of the weighted digraph $G$.

let $I_d$ denotes the $d \times d$ identity matrix.

let $Z^*$ denotes the $n \times d$ matrix that represent the relative coordinates of the desired formation.

## 3.2 Update Rule

The update rule (equation 11 on the formation stabilization paper[1]) for the position of each agent in global frame is:

$$p_i(l+1) = (1-h)p_i(l) + h\frac{1}{d_i}(\sum_{j \neq i} a_{ij}p_j(l) + Rb_i) \tag{1}$$

Such that :

- $l$ is the current time frame.
- $p_i(l)$ is the position of agent i in the current time frame.
- $h$ is the rate in the range [0,1] at which the agents converge to the formation.
- $d_i$ is the ith diagonal element of $D_{out}(G)$
- $a_{ij}$ is the element on the i-th row and j-th column of $A(G)$.
- $R$ is a $d \times d$ rotation matrix.
- $b$ is a $n \times d$ matrix define by equation 8 on the formation stabilization paper[1] as:

$$b = (b_1, ..., b_n) = (L(G) \otimes I_d)Z^* \tag{2}$$

## 4 Methodology

In this section, we go over the rationale for test parameters used the formation stabilization algorithm, the system setup, the teleoperation commands and visualization of the results.

### 4.1 Rationale for Test Parameters

We set $d = 3$ because we want to incorporate both the TurtleBots and the AR drones, so the system can be applied to 3D space.

We let $n = \mathbb{Z}_{\geq 2}$ where the exact number of the agents is dynamically specified by the user at run time. We need at least 2 agents to form a formation, thus the minimum number of agents allowed is 2.

We set $D_{out}(G) = I_n$ to achieve a full connected graph where any agent is affected by all the other agents.

We set

$$A(G) \in \mathbb{R}^{n \times n} = \begin{bmatrix} 0 & 1/n & ... & 1/n \\ 1/n & 0 & ... & 1/n \\ ... & ... & ... & ... \\ 1/n & 1/n & ... & 0 \end{bmatrix}$$

so each agent is equally affected by all the other agents.

We set

$$Z_{line} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 3 & 0 \\ 0 & 3 & 3 \\ 3 & 0 & 0 \\ 3 & 0 & 3 \\ 3 & 3 & 0 \\ 3 & 3 & 3 \end{bmatrix} \quad Z_{cube} = \begin{bmatrix} 0 & 0 & 0 \\ 0.5 & 0.5 & 0 \\ 1 & 1 & 0 \\ 1.5 & 1.5 & 0 \\ 2 & 2 & 0 \\ 2.5 & 2.5 & 0 \\ 3 & 3 & 0 \\ 3.5 & 3.5 & 0 \end{bmatrix} \quad Z_{square} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0.3 & 0 & 0 \\ 0.3 & 0.3 & 0 \end{bmatrix} \quad Z_{triangle} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0.3 & 0 & 0 \end{bmatrix}$$

which produce a cube formation and a line formation respectively, the cube formation would test if the algorithm works on a 3D formation, and the line formation can be use the test formations with any number of agents between 2 and 8 inclusive.

We set $h$ to 0.25 for a smooth yet timely convergence.

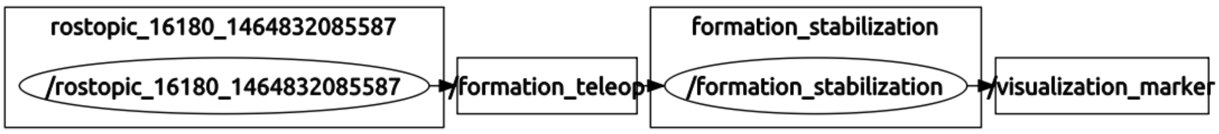
Lastly, We set $R = R_z(\theta_z)R_y(\theta_y)R_x(\theta_x)$ where

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \quad R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad R_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

as the 3D rotation matrices along different axes of rotation[6].

## 4.2 System Set Up

The system is set up in ROS framework and visualized in RVIZ. The control flow is quite straight forward. When the user send a command to the "/formation_teleop" topic, it triggers the callback method in the formation_stabilization node. This node is where the formation stabilization algorithm is implemented and the callback method would update the appropriate parameters in the algorithm and calculate the new goal positions of all the agents. Then, the goal positions are send to the "/visualization_maker" topic, which RVIZ uses to create the visualization of the positions in 3D space. Here is a graph that summarizes the simple workflow.



## 4.3 Teleoperation Commands

Now that we know how the system works, let's take a look at the input of the system–the teleoperation commands that the user can send to control the formation of the agents.

Availiable opcodes are:

(0-9) changes the number of agents

(+) adds an agent

(-) removes an agent

(c) changes the formation to a cube (8 agents)

(h) shows all availialbe opcodes (this list)

(i) collapses the formation inward

(l) changes the formation to a line (2+ agents)

(o) spreads the formation outward

(t) changes the formation to a triangle (3 agents)

(x) changes the axis of rotation to x-axis

(y) changes the axis of rotation to y-axis

(z) changes the axis of rotation to z-axis

(s) changes the formation to a square (4 agents)

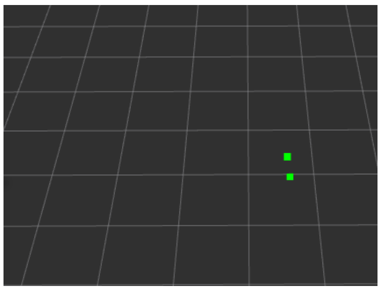
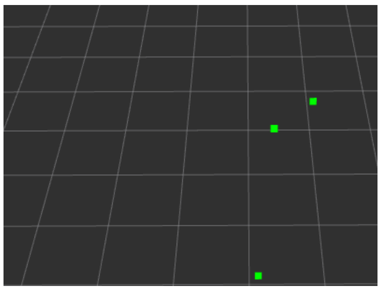
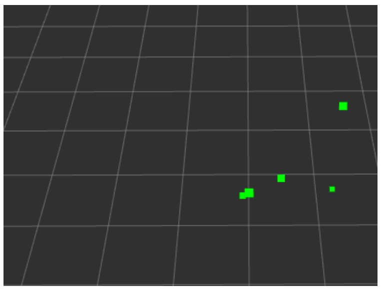
(<) rotates the formation counter-clockwise

(>) rotates the formation clockwise

## 4.4 Teleoperation Visualizations

### 4.4.1 Add/Remove Agent(s)

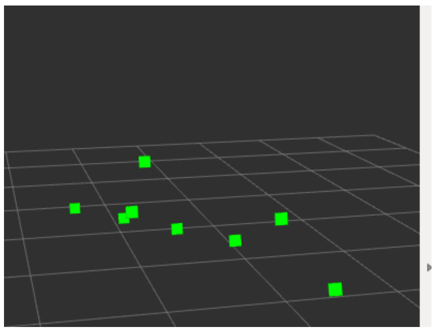
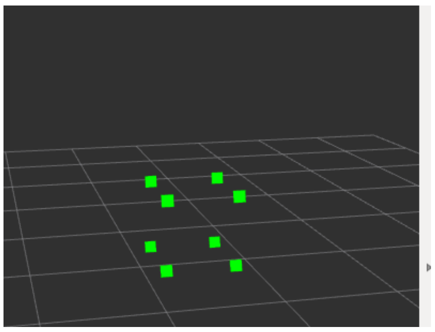
The simulation always starts with 2 agents because that's the minimum number of agents need to make a formation. We will start off by adding an agent using the (+) operation, then we change the number of agents to 5, using (5) operation.

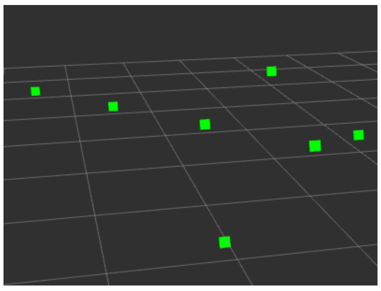
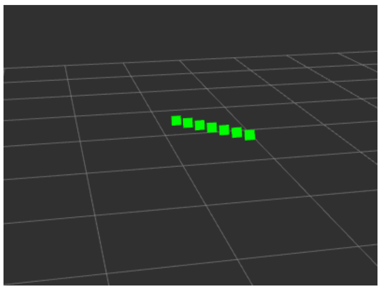
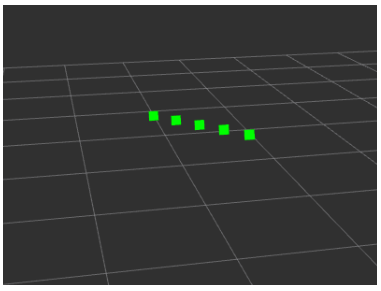
We noticed that the position of the original agents are not preserved because adding an agent, which spawns randomly on the map, disturbs the stability of the formation thus causing the existing agents to reposition at an new equilibrium. Removing an agents suffers from the same problem. However, for both cases the disturbance can be prevented by immediately sending a stable formation that utilizes all the available agents.

### 4.4.2 Formations

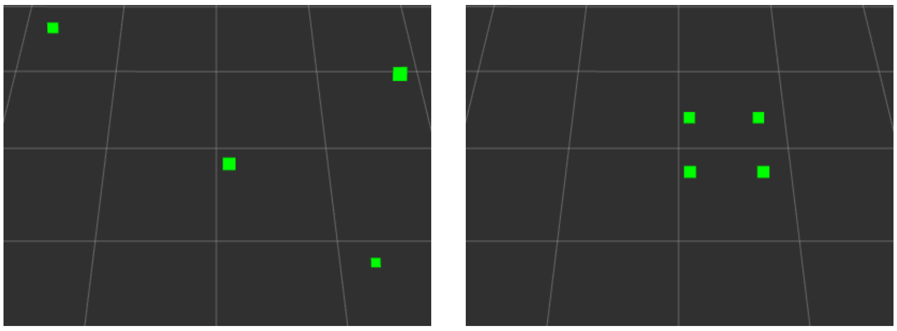
Cube Formation, which requires strictly 8 available agents. We start with 8 agents at arbitrary positions, and when the user send the (c) command, the formation converge into a cube.

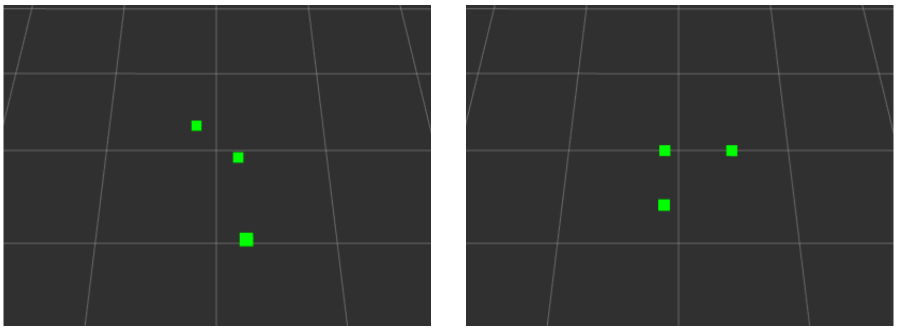
Line Formation, which can be form with 2 - 8 agents. (Also can be formed by more than 8 agents if the length of configuration matrix $Z_{line}$ is extended.) The images below shows 7 agents at arbitrary positions, and then the line formation command (l) was given by the user. lastly, the number of agents were reduced to 5.

Square Formation, which requires strictly 4 available agents:

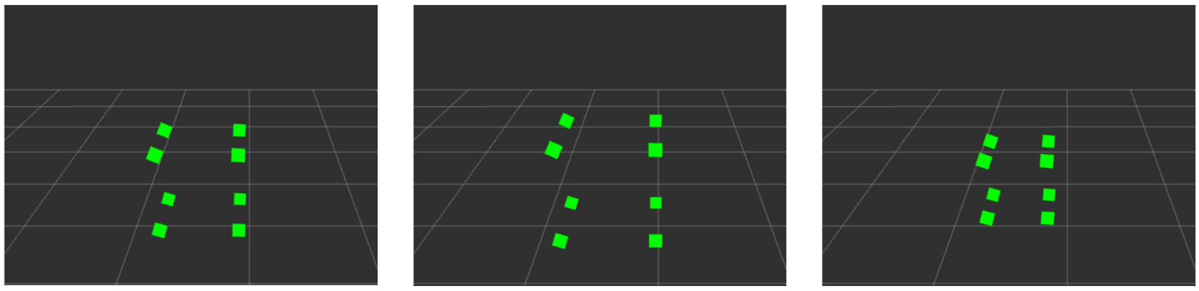


Triangle Formation, which requires strictly 3 available agents:



### 4.4.3 Scaling

The formation size can be scaled so that the agents are more clustered together or more spread out. In the following images, we see an cube formation at 100%, 120%, and 80% scaling.



### 4.4.4 Rotation

To rotate an formation in 3D, we need to let the system know which axis to rotate about. Users can specify the axis of rotation by sending one of (x, y, z) teleop command.

I realized is difficult to demonstrate 3D rotation with flat images, so I created a short video, you can see the video demo on `https://youtu.be/6oGa8_5gdJI`

# 5 Tips

## 5.1 For Formation Stabilization Algorithm

A solid understanding of paper "Global and robust formation-shape stabilization of relative sensing networks"[1] by Prof. Cortés is very helpful. To understanding the paper, I highly recommend some background in graph theory and matrix operation. Digraph[7], Laplacian matrix[8], and Kronecker product[9] are especially helpful in understanding the b-matrix2 in the update algorithm we used for the implementation.

### 5.2 For ROS

Going through the beginner's tutorials[10] for ROS will help with understanding the structure of ROS. The particular tutorials that should be attempt before modifying this project are #1-8, 12, 13, 15, 16 int the beginner's tutorials section. Understanding ROS rqt_graph (covered in tutorial #8) can help with understanding the graphs in section 4.3. The TurtleSim Tutorial[?] will help with understanding how Turtlesim simulation works.

### 5.3 For Rotation Matrices

To calculate the rotation of the formation in 3D, use this formula $R = R_z(\theta_z)R_y(\theta_y)R_x(\theta_x)$. ($R_z, R_y, R_x$ can be found in Section 4.1) The Wikipedia page[6] offers enough information to understand the 3D rotational matrices.

## 6 Pitfalls

The algorithm does a great job at stabilizing the agents into the desired formation, but there is the chance for collision when transitioning from an initial state into the formation. It is not an issue with simulation, but for testing on physical machines, we would need to address the collision issue during transition.

When the user dynamically increase or decrease the number of agents, the stability would be lost, thus causing random behaviors in all the agents. This can be avoid by sending an appropriate formation right after changing the number of agents.

## 7 Conclusion

With the formation stabilization algorithm and some simple teleoperation commands, a network of agents be can controls control by a single user. The system is implemented in ROS framework and visualized in RVIZ. The user can send a simple teleop command, and the agents would move accordingly in 3D space while maintaining their formation. These operations includes changing formation shape, scaling formation size and rotating formation. Using RVIZ, we visually confirm that the system gives user the command over multiple agents. The next step is to combine this path planning system with our location and motion planning subsystem to produce a full system that we can test on the TurtleBots and AR-Drones.

## References

[1] Jorge Cortés *Global and robust formation-shape stabilization of relative sensing networks.* Automatica 45 (12) (2009), 2754-2762 `http://carmenere.ucsd.edu/jorge/publications/data/2008_Co-auto.pdf`

[2] *Multi-Agent Robotics Lab* `http://muro.ucsd.edu/`

[3] *TurtleBot* `http://www.turtlebot.com/`

[4] *Ros* `http://www.ros.org/`

[5] *Parrot AR Drone* `http://www.parrot.com/usa/products/ardrone-2/`

[6] *Rotation matrix* `https://en.wikipedia.org/wiki/Rotation_matrix`

[7] *Directed Graph* `https://en.wikipedia.org/wiki/Directed_graph`

[8] *Laplacian matrix* `https://en.wikipedia.org/wiki/Laplacian_matrix`

[9] *Kronecker product* `https://en.wikipedia.org/wiki/Kronecker_product`

[10] *Ros Tutorials* `http://wiki.ros.org/ROS/Tutorials`

[11] *UCSD ROS Project Repository* `https://github.com/evangravelle/ucsd_ros_project/`

[12] *Project Source Code* `https://github.com/evangravelle/ucsd_ros_project/blob/formation_stabilization/deployment/formation_stabilization/scripts/fs3d_node.py`