
上海交通大学

SHANGHAI JIAO TONG UNIVERSITY



多机器人系统与控制

一致性协议在不同图结构下的结果差异

报告名称: 多机器人系统与控制实验一报告

小组编号: 第二组

组员姓名: 陈炜昊 王喆隆

易子淇 章雨悠

指导教师: 熊振华 董伟 吴建华

2021 年 12 月

一致性协议在不同图结构下的结果差异

实验目的和要求

基本任务

1. 在示例代码（角度一致性）的基础上，通过改变图结构，分析一致性协议在不同图结构下的结果差异。
2. 通过一致性协议使小车编队收敛在一条直线上。

附加任务

1. 能同时控制角速度和速度，而不是先转再走的分离控制。
2. 均可无碰撞地收敛到一条直线上。

实验方案

一致性收敛

任务一中角度一致性的控制，采用如下动力学方程进行控制

$$\mathbf{u}_\theta = \dot{\boldsymbol{\theta}} = -L\boldsymbol{\theta}$$

同理任务二也可以采用类似的控制律，假设需要收敛的直线表达式为 $y = \text{Const}$ （即与 x 轴平行的直线）

$$\mathbf{u}_y = \dot{\mathbf{y}} = -L\mathbf{y}$$

角度+位置一致性

为了同时控制角速度和速度，而不是先转再走的分离控制。我们将二者的一致性协议进行融合，设计控制律如下

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_y \\ \mathbf{u}_\theta \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{y}} \\ \dot{\boldsymbol{\theta}} \end{bmatrix} = -L \begin{bmatrix} \mathbf{y} \\ \boldsymbol{\theta} \end{bmatrix}$$

考虑到实际程序控制中，控制量为小车线速度与角速度，即 $\mathbf{u} = [v, w]$ ，无法直接控制 $\dot{\mathbf{y}}$ 。此外小车的 θ 与目标方向可能存在偏差，为了使小车在偏离目标方向时尽可能慢速以便于及时调整角度，需要修改控制律如下

$$\mathbf{u} = \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{y}} \times \cos\left(\theta - \frac{\pi}{2}\right) \\ \dot{\boldsymbol{\theta}} \end{bmatrix} = -L \begin{bmatrix} \mathbf{y} \times \cos\left(\theta - \frac{\pi}{2}\right) \\ \boldsymbol{\theta} \end{bmatrix}$$

无碰撞收敛

为了规避碰撞等意外情况，我们设计了碰撞检测环节，初定方案是判断与邻居之间的距离，如果过近则会背向远离。考虑到实际实验的场地较小，我们采用了优化后的碰撞检测方案，如果其他机器人出现在当前机器人行进方向上且两者距离相近，则将线速度置零，先将角度调整合适后继续行进，数学表达如下

$$\text{Condition 1} \quad \left| \arctan \left(\frac{y_j - y_i}{x_j - x_i} \right) - \theta_i \right| < \varepsilon_1$$

$$\text{Condition 2} \quad \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} < \varepsilon_2$$

$$\text{Then} \quad v_i = 0, \quad w_i = -(\mathbf{L}\boldsymbol{\theta})_i$$

目标直线拟合及一致性

在仿真测试中，我们发现如果机器人初始位置极端，则收敛到表达式为 $y = \text{Const}$ 的直线用时较长，因此我们期望每台机器人在运动过程中实时分布式拟合目标直线。初定方案是利用机器人及其邻居的位置进行最小二乘法拟合，得到直线方程如下

$$ax + by - c = 0$$

并计算机器人与目标直线的距离设计控制律

$$\delta = \frac{|ax_i + by_i - c|}{\sqrt{a^2 + b^2}}$$

$$\phi = \arctan \left(-\frac{a}{b} \right)$$

$$\mathbf{u} = \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} \dot{\delta} \times \cos(\theta - \phi - \frac{\pi}{2}) \\ \dot{\theta} \end{bmatrix} = -\mathbf{L} \begin{bmatrix} \delta \times \cos(\theta - \phi - \frac{\pi}{2}) \\ \theta \end{bmatrix}$$

但在仿真测试中发现，基于最小二乘拟合得到的直线无法收敛，独立个体有可能会选择不同的目标直线（例如初始位置呈星形分布的情况）。因此我们更换了直线拟合方法，首先利用一致性协议得到全局机器人的角度均值，而后将与之垂直的角度斜率确定为目标直线斜率，进而以最小化各机器人运动代价为目标优化求解得到直线截距。数学描述如下

$$\begin{aligned} \xi(0) &= \boldsymbol{\theta} \\ \dot{\xi} &= -\mathbf{L}\xi \\ \implies \xi &\rightarrow \frac{(\mathbf{1}^T \xi(0))\mathbf{1}}{N}, \quad t \rightarrow \infty \end{aligned}$$

上式最终得到的值即为各机器人角度均值

$$\begin{aligned} k &= \tan \left(\xi_{t \rightarrow \infty} + \frac{\pi}{2} \right) \\ \min \sum_{j \in N_i} |y_j - kx_j - b_i| &\implies b_i \end{aligned}$$

得到直线方程后利用与初定方案相同的控制器即可实现收敛。

数据处理及分析

任务一

首先运行示例代码（全连通图），得到程序收敛时间为 5.62s，且收敛情况良好，实验视频见 `../video/angle_1.mp4`

而后在保证连通性的前提下改变图结构，减小拉普拉斯矩阵的 λ 值，设计如下

$$L = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{bmatrix} \quad \text{特征值} \lambda = \begin{bmatrix} 0 \\ 0.8299 \\ 2.6889 \\ 4 \\ 4.4812 \end{bmatrix}$$

最终程序收敛时间为 9.20s，与理论结果一致，图结构的 λ_2 越小，系统收敛速度越慢。此时收敛误差也较大，实验视频见 ../video/angle_2.mp4

误差分析 可能是程序判定收敛条件 $|\delta_\theta| = |\dot{\theta}| = | -L\theta | < \varepsilon$ 中的 ε 较大，而部分节点邻居数量又少，误差累积导致。

任务二

收敛至直线 $y = Const$ 的视频为见 ../video/line_1.mp4

实时拟合表达式的直线收敛视频见 ../video/line_2.mp4

视频中本组实现并完成了实验的所有任务，也不难发现当小车位置识别不稳定（画面中小车上方坐标系不停闪动）时对收敛的情况有一定影响，但在位置识别重新稳定后编队收敛重新正常且表现良好。

附录

实验视频

视频见 video 目录

line_1代码

```
/*
 * Date: 2021-11-30
 * Description: To a line y = Const
 * Group: 2
 */
#include <swarm_robot_control.h>

int main(int argc, char** argv) {
    ros::init(argc, argv, "swarm_robot_control_formation");
    ros::NodeHandle nh;
    tf::TransformListener tf_listener;

    /* Initialize swarm robot */
    for(int index = 0; index < swarm_robot_id.size(); index++) {
        std::string vel_topic = "/robot_" +
std::to_string(swarm_robot_id[index]) + "/cmd_vel";
        swarm_robot_cmd_vel_pub[index] = nh.advertise<geometry_msgs::Twist>
(vel_topic, 10);
    }

    /* Set L Matrix*/
    Eigen::MatrixXd lap(swarm_robot_id.size(), swarm_robot_id.size());
    lap << 4, -1, -1, -1, -1,
        -1, 4, -1, -1, -1,
        -1, -1, 4, -1, -1,
        -1, -1, -1, 4, -1,
        -1, -1, -1, -1, 4;
```

```

/* Convergence threshold */
double conv_th = 0.02; // Threshold of angle, in rad
double conv_dis = 0.02; // Threshold of distance

/* Velocity scale and threshold */
double MAX_W = 1; // Maximum angle velocity (rad/s)
double MIN_W = 0.05; // Minimum angle velocity(rad/s)
double MAX_V = 0.2; // Maximum linear velocity(m/s)
double MIN_V = 0.01; // Minimum linear velocity(m/s)
double k_w = 0.1; // Scale of angle velocity
double k_v = 0.1; // Scale of linear velocity

/* Mobile robot poses and for next poses */
Eigen::VectorXd cur_x(swarm_robot_id.size());
Eigen::VectorXd cur_y(swarm_robot_id.size());
Eigen::VectorXd cur_theta(swarm_robot_id.size());
Eigen::VectorXd del_x(swarm_robot_id.size());
Eigen::VectorXd del_y(swarm_robot_id.size());
Eigen::VectorXd del_theta(swarm_robot_id.size());

/* Get swarm robot poses firstly */
std::vector<std::vector<double> > current_robot_pose(swarm_robot_id.size());
std::vector<bool> flag_pose(swarm_robot_id.size(), false);
bool flag = false;
while(! flag) {
    flag = true;
    for(int i = 0; i < swarm_robot_id.size(); i++) {
        flag = flag && flag_pose[i];
    }
    for(int i = 0; i < swarm_robot_id.size(); i++) {
        std::vector<double> pose_robot(3);
        if(getGazeboRobotPose(i, pose_robot, tf_listener)) {
            current_robot_pose[i] = pose_robot;
            flag_pose[i] = true;
        }
    }
}
ROS_INFO_STREAM("Succeed getting pose!");
for(int i = 0; i < swarm_robot_id.size(); i++) {
    cur_theta(i) = current_robot_pose[i][2];
}
for(int i = 0; i < swarm_robot_id.size(); i++) {
    cur_y(i) = current_robot_pose[i][1];
}
for(int i = 0; i < swarm_robot_id.size(); i++) {
    cur_x(i) = current_robot_pose[i][0];
}

/* Convergence sign */
bool is_conv = false; // Convergence sign of angle
/* While loop */
while(! is_conv) {
    /* Judge whether reached */
    del_y = -lap * cur_y;
    del_theta = -lap * cur_theta;
    is_conv = true;
    for(int i = 0; i < swarm_robot_id.size(); i++) {

```

```

        if(std::fabs(del_y(i)) > conv_dis || std::fabs(del_theta(i)) >
conv_th) {
            is_conv = false;
        }
    }

    /* Swarm robot move */
    for(int i = 0; i < swarm_robot_id.size(); i++) {
        bool crash_flag = false;
        double v = del_y(i) * k_v;
        double w = del_theta(i) * k_w;
        for (int j = 0; j < swarm_robot_id.size(); j++){
            if (j == i){
                continue;
            }
            // 判断是否有其他机器人在行进方向上
            if (std::fabs((cur_y(j) - cur_y(i))/(cur_x(j) - cur_x(i)) -
cur_theta(i)) < 0.05 &&\
                sqrt((cur_x(j) - cur_x(i))*(cur_x(j) - cur_x(i)) + (cur_y(j) -
cur_y(i))*(cur_y(j) - cur_y(i))) < 1){
                v = 0;
            }
        }
        if (cur_theta(i) < 0){
            v = -v;
        }
        v = checkVel(v, MAX_V, MIN_V);
        w = checkVel(w, MAX_W, MIN_W);
        moveRobot(i, v, w);
    }

    /* Time sleep for robot move */
    ros::Duration(0.05).sleep();

    /* Get swarm robot poses */
    for(int i = 0; i < swarm_robot_id.size(); i++) {
        std::vector<double> pose_robot(3);
        if(getGazeboRobotPose(i, pose_robot, tf_listener)) {
            current_robot_pose[i] = pose_robot;
        }
    }
    for(int i = 0; i < swarm_robot_id.size(); i++) {
        cur_theta(i) = current_robot_pose[i][2];
    }
    for(int i = 0; i < swarm_robot_id.size(); i++) {
        cur_y(i) = current_robot_pose[i][1];
    }
    for(int i = 0; i < swarm_robot_id.size(); i++) {
        cur_x(i) = current_robot_pose[i][0];
    }
}

/* Stop all robots */
stopRobot();
ROS_INFO_STREAM("Succeed!");
return 0;
}

```

line_2代码

```
/*
 * Date: 2021-11-30
 * Description: To a fitted line
 * Group: 2
 */
#include <swarm_robot_control.h>

int main(int argc, char** argv) {
    ros::init(argc, argv, "swarm_robot_control_formation");
    ros::NodeHandle nh;
    tf::TransformListener tf_listener;

    /* Initialize swarm robot */
    for(int index = 0; index < swarm_robot_id.size(); index++) {
        std::string vel_topic = "/robot_" +
std::to_string(swarm_robot_id[index]) + "/cmd_vel";
        swarm_robot_cmd_vel_pub[index] = nh.advertise<geometry_msgs::Twist>
(vel_topic, 10);
    }

    /* Set L Matrix*/
    Eigen::MatrixXd lap(swarm_robot_id.size(), swarm_robot_id.size());
    lap <<  4, -1, -1, -1, -1,
           -1, 4, -1, -1, -1,
           -1, -1, 4, -1, -1,
           -1, -1, -1, 4, -1,
           -1, -1, -1, -1, 4;

    /* Convergence threshold */
    double conv_th = 0.05; // Threshold of angle, in rad
    double conv_dis = 0.02; // Threshold of distance

    /* velocity scale and threshold */
    double MAX_W = 1; // Maximum angle velocity (rad/s)
    double MIN_W = 0.05; // Minimum angle velocity(rad/s)
    double MAX_V = 0.1; // Maximum linear velocity(m/s)
    double MIN_V = 0.01; // Minimum linear velocity(m/s)
    double k_w = 0.1; // Scale of angle velocity
    double k_v = 0.1; // Scale of linear velocity

    /* Mobile robot poses and for next poses */
    Eigen::VectorXd cur_x(swarm_robot_id.size());
    Eigen::VectorXd cur_y(swarm_robot_id.size());
    Eigen::VectorXd cur_theta(swarm_robot_id.size());
    Eigen::VectorXd del_x(swarm_robot_id.size());
    Eigen::VectorXd del_y(swarm_robot_id.size());
    Eigen::VectorXd del_theta(swarm_robot_id.size());
    Eigen::VectorXd del_dis(swarm_robot_id.size());

    /* Get swarm robot poses firstly */
    std::vector<std::vector<double>> current_robot_pose(swarm_robot_id.size());
    std::vector<bool> flag_pose(swarm_robot_id.size(), false);
    bool flag = false;
    while(! flag) {
        flag = true;
```

```

    for(int i = 0; i < swarm_robot_id.size(); i++) {
        flag = flag && flag_pose[i];
    }
    for(int i = 0; i < swarm_robot_id.size(); i++) {
        std::vector<double> pose_robot(3);
        if(getGazeboRobotPose(i, pose_robot, tf_listener)) {
            current_robot_pose[i] = pose_robot;
            flag_pose[i] = true;
        }
    }
}

ROS_INFO_STREAM("Succeed getting pose!");
for(int i = 0; i < swarm_robot_id.size(); i++) {
    cur_theta(i) = current_robot_pose[i][2];
}
for(int i = 0; i < swarm_robot_id.size(); i++) {
    cur_y(i) = current_robot_pose[i][1];
}
for(int i = 0; i < swarm_robot_id.size(); i++) {
    cur_x(i) = current_robot_pose[i][0];
}

/* Convergence sign */
bool is_conv = false;    // Convergence sign
/* While loop */
while(! is_conv) {
    /* 拟合直线 */
    Eigen::Matrix<double, 3, 1> fitline(swarm_robot_id.size(), 3);
    for (int i = 0; i < swarm_robot_id.size(); i++){
        double k_s = 0;
        int count = 0;
        for (int j = 0; j < swarm_robot_id.size(); j++){
            if (j == i){
                continue;
            }
            else {
                k_s += tan(cur_theta(i));
                ++count;
            }
        }
        k_s /= count;
        fitline(i, 0) = 1/k_s; fitline(i, 1) = 1;
        for (int j = 0; j < swarm_robot_id.size(); j++){
            if (j == i){
                continue;
            }
            else {
                fitline(i, 2) += fitline(i, 0)*cur_x(j) + fitline(i,
1)*cur_y(j);
            }
        }
        fitline(i, 2) /= count;
        del_dis(i) = fitline(i, 0)*cur_x(i) + fitline(i, 1)*cur_y(i) -
fitline(i, 2);
    }

    /* Judge whether reached */
    del_theta = -lap * cur_theta;

```



```

        is_conv = true;
        for(int i = 0; i < swarm_robot_id.size(); i++) {
            if(std::fabs(del_dis(i)) > con_dis || std::fabs(del_theta(i)) >
conv_th) {
                is_conv = false;
            }
        }

        /* Swarm robot move */
        for(int i = 0; i < swarm_robot_id.size(); i++) {
            bool crash_flag = false;
            double v = del_dis(i) * k_v;
            double w = del_theta(i) * k_w;
            for (int j = 0; j < swarm_robot_id.size(); j++){
                if (j == i){
                    continue;
                }
                if (std::fabs((cur_y(j) - cur_y(i))/(cur_x(j) - cur_x(i)) -
cur_theta(i)) < 0.05 &&\
                    sqrt((cur_x(j) - cur_x(i))*(cur_x(j) - cur_x(i)) + (cur_y(j) -
cur_y(i))*(cur_y(j) - cur_y(i))) < 1){
                    v = 0;
                }
            }
            double del = atan(-fitline(i, 0)/fitline(i, 1));
            if (cos(del + 3.1415926/2 - cur_theta(i)) > 0){
                v = -v;
            }
            v = checkVel(v, MAX_V, MIN_V);
            w = checkVel(w, MAX_W, MIN_W);
            moveRobot(i, v, w);
        }
        /* Time sleep for robot move */
        ros::Duration(0.05).sleep();

        /* Get swarm robot poses */
        for(int i = 0; i < swarm_robot_id.size(); i++) {
            std::vector<double> pose_robot(3);
            if(getGazeboRobotPose(i, pose_robot, tf_listener)) {
                current_robot_pose[i] = pose_robot;
            }
        }
        for(int i = 0; i < swarm_robot_id.size(); i++) {
            cur_theta(i) = current_robot_pose[i][2];
        }
        for(int i = 0; i < swarm_robot_id.size(); i++) {
            cur_y(i) = current_robot_pose[i][1];
        }
        for(int i = 0; i < swarm_robot_id.size(); i++) {
            cur_x(i) = current_robot_pose[i][0];
        }
    }
    /* Stop all robots */
    stopRobot();
    ROS_INFO_STREAM("Succeed!");
    return 0;
}

```