Multi-robot Coordination

A project of the 2020 Robotics Course of the School of Information Science and Technology (SIST) of ShanghaiTech University

Liu Wenji (liuwj@shanghaitech.edu.cn)
Sun yiran (sunyr@shanghaitech.edu.cn)

Beta version, January 2021

Abstract

In the field of mobile robotics, the research of multi-robot systems has grown significantly in recent years. Here we use several Turtle bots to implement the multi-robot coordination task.We build 4 Turtle bots,and focus on multi-robot coordination which involve the localization and navigation of Turtle bots. Non-holonomic optimal reciprocal collision avoidance (NH-ORCA) will be implemented for navigation. A series of multi-robot coordination results and system evaluation methods are given at the end of the paper.

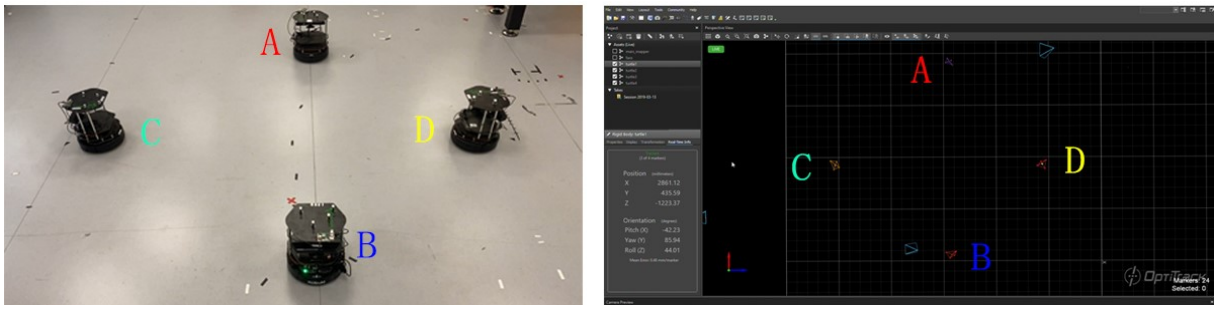


Figure 1: our system and results

1 Introduction

Multi-robot systems are an important part of robotics research.Having made great progress in the development of the basic problems concerning single-robot control, many researchers shifted their focus to the study of multi-robot coordination. Most of today's robots fall into one of three primary categories, namely manipulators, mobile robots and humanoid robots. This paper focuses on multiple mobile robot systems (MMRSs), in which robots should work together to accomplish a given task by moving around in the environment.

Here we want to build a multi-robot system ,which use local WIFI to communicates with each other and our goal is to complete some path-following tasks. We focus on multi-robot coordination which involve the localization and navigation of Turtle bots. Additionally The total system consists of 4 turtle bots and corresponding tracking system.Besides the construction of the multi-robot system, we decide to test the NH-ORCA algorithm.

First of all, on setting up the turtle robot, we need to add some additional functionalities to the robot such as micro-computer and wifi support for communicating and executing algorithms.Then

we propose 3 options, but we'd rather take the first option which markers in STAR Labs' tracking system ,as it's sophisticated and more suitable for a quick test.Finally,we decide to test the NH-ORCA algorithm. A series of multi-robot coordination results and system evaluation methods are given at the end of the paper.

The remainder of the paper is organized as follows:

Section 2 reviews literature and open-source-ROS packages relevant to this project; Section 3 describe our idea and proposed system and algorithm in detail; Section 4 describes how to test our systems; the paper is concluded with a discussion in Section 5.

2 State of the art

2.1 Related Papers

A single-robot system contains only one individual robot that is able to model itselfthe environment and their interaction.A multi-robot system contains more than one individual robot, whether group homogeneous or heterogeneous. [YJC13] presents a systematic survey and analysis of the existing literature on coordination, especially in multiple mobile robot systems (MMRSs). A series of related problems have been reviewed, which include a communication mechanism, a planning strategy and a decision-making structure.

This paper [Fox+00] presents a statistical algorithm for collaborative mobile robot localization. His approach uses a sample-based version of Markov localization, capable of localizing mobile robots in an any time fashion. When teams of robots localize themselves in the same environment, probabilistic methods are employed to synchronize each robot's belief whenever one robot detects another. As a result, the robots localize themselves faster, maintain higher accuracy, and high-cost sensors are amortized across multiple robot platforms.

Automated vehicle coordination can be used to control vehicles across traffic intersections safely and efficiently. The following two paper did some work in this field. [Jia+17] proposes a novel parallelizable algorithm, which solves the coordination problem at traffic intersections under a given precedence order by using a tailored variant of the augmented Lagrangian based alternating direction inexact Newton method (ALADIN). Here, each vehicle solves its own optimal control problem and exchanges information about arrivial and departure times at the intersection with its neighbors such that collisions are avoided. **[shi2018]** proposes a novel parallelizable algorithm, which solves the coordination problem at traffic intersections under a given precedence order by using a tailored variant of the augmented Lagrangian based alternating direction inexact Newton method (ALADIN). Here, each vehicle solves its own optimal control problem and exchanges information about arrivial and departure times at the intersection with its neighbors such that collisions are avoided. They all illustrate the performance of ALADIN by analyzing two scenarios, one during rush hour and one at low-traffic conditions.

Multi-robot coordination is the core task of multi robot systems. The overall system performance can be directly affected by the quality of coordination and control. Coordination can be static or dynamic. Static coordination (also known as deliberative coordination [INS00] or offline coordination [TRS00]) generally refers to the adoption of a convention prior to engaging in the task. For example, some rules in traffic control problems include "keep right", "stop at intersection" and "keep sufficient space between yourself and the robot in front of you" [KNT92]. Dynamic coordination(also known as reactive coordination [INS00] or online coordination [TRS00])) occurs during the execution of a task, and is generally based on the analysis and synthesis of information. The information can be obtained through the means of communication.

Deliberation and reactivity are important features of a robotic system: both in the case of a single robot and in a robotic team. [INS00] have focused they attention on the issue of reactivity and social deliberation, by distinguishing the single robot from the MRS setting, and by proposing a

characterization of MRS in terms of reactivity and social deliberation. In this paper,they first recall the architectural issues that have been addressed for a single robot in order to develop a reactive and/or a deliberative behavior. Then, they characterize which behavior is expected from a reactive or a deliberative MRS. Finally, they discuss reactivity and social deliberation with respect to the taxonomy introduced in the previous section.

Communication, as a means of coordination, often emerges as a rational behaviour in multi-robot environments. In fact, the communication is a mode of interaction between robots. [CFK97] classified the communication structure into three types according to the mode of interaction, which includes: interaction via the environment, interaction via sensing and interaction via explicit communications. [FIN04] distinguished two different types of communication depending on the way in which the robots exchange information, which includes direct and indirect communication.

2.2 Current Packages and Softwares

We will focus on four parts which support separate functionalities in the project.

- Single (Turtle2) Robot Programming and Control
- ROS Localization/Navigation Tool
- C++ Non-linear optimization Tool/ Optimal Control Tool

Though we may use the lab's tracking system for localization, instead of using ROS's localization/navigation tool, it still worth a check.

Since we may have a new multi-MPC algorithm to implement,we need a C++ Optimization solver to deal with the constraints functions and find solutions.

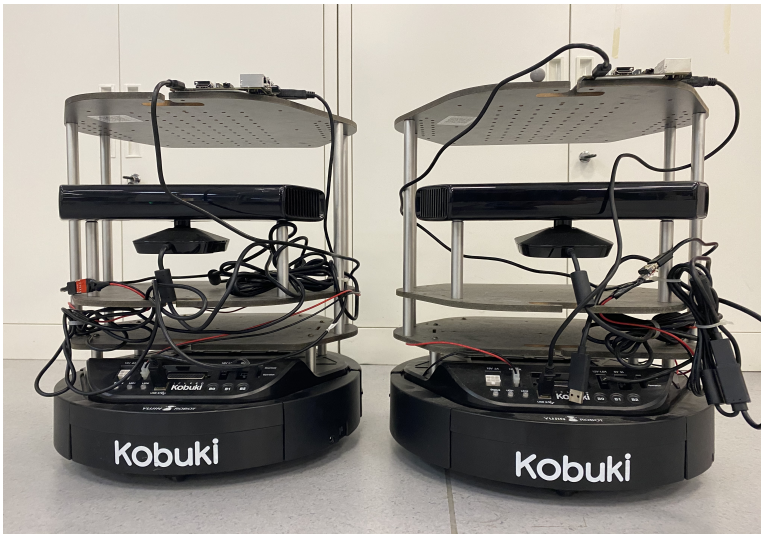


Figure 2: image of our robot:Kobuki

2.2.1 Turtle2 Robot Programming and Control

We find multiple sources of tutorials on turtlebot which should be helpful: `http://wiki.ros.org/Robots/TurtleBot` and `https://www.turtlebot.com/learn/`.

1. install Tinker Board

2. install the DC/DC with power plugs

3. connect to our local WiFi

Here we give a detailed intro on this package.

The primary requirement for getting your TurtleBot running is connecting to a network so that you can operate it remotely.You can use the Network Manager in the upper right corner of your screen to connect to a WiFi network. After the network is started, it is time to set your ROS_MASTER_URI and $ROS\_HOSTNAME$. The ROS_MASTER_URI tells the rest of the nodes at which address they can find the ROS master .

```
#The localhost IP address = IP address for the Master node
$ export ROS_MASTER_URI=http://localhost:11311
#The IP address for the Master node
$ export ROS_HOSTNAME=192.168.8.100
```

Once the network is setup, we now need to bring up the TurtleBot software. This starts up the TurtleBot with the basic single master ROS environment in which all processes can be started/stopped via roslaunchers. Enter the following line in a terminal on the TurtleBot netbook:

```
$ ROS_NAMESPACE=turtleX roslaunch turtlebot_bringup minimal.launch
```

The following command allows you to tele-op your TurtleBot with a keyboard.

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

Also, we can write codes to send control messages to turtlebot, as referenced in here `http://edu.gaitech.hk/turtlebot/free-space-navigation.html`.

2.2.2 Turtle Bot Localization

First, you need to start the robot drivers and then start the ROS node responsible for building the map. It has to be noted that to build a map ROS uses the *gmapping* software package, that is fully integrated with ROS.

The gmapping package contains a ROS wrapper for OpenSlam's Gmapping. The package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called $slam_gmapping$. Using $slam_gmapping$, you can create a 2-D occupancy grid map (like a building floorplan) from laser and pose data collected by a mobile robot.

To start the robot, on the master node (the robot machine) run the following commands, on three terminals:

```
roscore
roslaunch turtlebot_bringup minimal.launch
roslaunch turtlebot_navigation gmapping_demo.launch
```

To drive the robot from your workstation, then, on your workstation (the host node), run the following commands:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
roslaunch turtlebot_teleop keyboard_teleop.launch
```

On the Navigation part, we can also write codes(http://edu.gaitech.hk/turtlebot/map-navigation.html) to do map-based navigation using the map built before.

2.2.3 C++ Non-linear optimization Tool/ Optimal Control Tool

For optimal control tools, we introduces two packages:***PSOPT*** and ***OpenOCL***.

PSOPT is an open source optimal control software package written in C++ that uses direct collocation methods, including pseudospectral and local discretizations. Pseudospectral methods solve optimal control problems by approximating the time-dependent variables using global polynomials, such as Legendre or Chebyshev functions. Local discretization methods approximate the time dependent functions using local splines, and can be seen as implementations of implicit Runge-Kutta integrators. With both global and local methods, differential equations, continuous constraints and integrals associated with the problem are discretized over a grid of nodes. Sparse nonlinear programming is then used to find local optimal solutions.

With ***OpenOCL*** we can solve a large class of optimal control problems including non-linear, continuous-time, multi-stage, and constrained problems, which can appear in the context of trajectory optimization and model predictive control. The types of dynamical systems that are supported are all systems that can be described by ordinary differential equations or differential algebraic equations.

For nonlinear constraint solvers, we introduces ***OptimLib*** and ***NLopt***.

OptimLib is a lightweight C++ library of numerical optimization methods for nonlinear functions. Features:

- A C++11 library of local and global optimization algorithms, as well as root finding techniques
- Derivative-free optimization using advanced, parallelized metaheuristic methods
- Constrained optimization routines to handle simple box constraints, as well as systems of nonlinear constraints
- OptimLib supports the following templated linear algebra libraries: Armadillo Eigen
- OpenMP-accelerated algorithms for parallel computation
- Straightforward linking with parallelized BLAS libraries, such as OpenBLAS
- Available as a header-only library, or as a compiled shared library

.

NLopt is a free/open-source library for nonlinear optimization, providing a common interface for a number of different free optimization routines available online as well as original implementations of various other algorithms. Its features include:

- Support for large-scale optimization (some algorithms scalable to millions of parameters and thousands of constraints).
- Both global and local optimization algorithms.
- Algorithms using function values only (derivative-free) and also algorithms exploiting user-supplied gradients.
- Algorithms for unconstrained optimization, bound-constrained optimization, and general nonlinear inequality/equality constraints.

In all cases, we recommend to use OptimLib as it's easy to complie and directly supports data structure like Eigen from ROS.

3 System Description

Here we want to build a multi-robot system which communicates with each other and complete some path-following tasks. The total system consists of 4 turtle bots and corresponding tracking system. Besides the construction of the multi-robot system, we decide to test the NH-ORCA algorithm.

Necessary steps to complete this project has been discussed and listed in below subsections.

3.1 Robot setup

On setting up the turtle robot, we need to add some additional functionalities to the robot such as micro-computer and wifi support for communicating and executing algorithms. This mainly contains 3 steps:

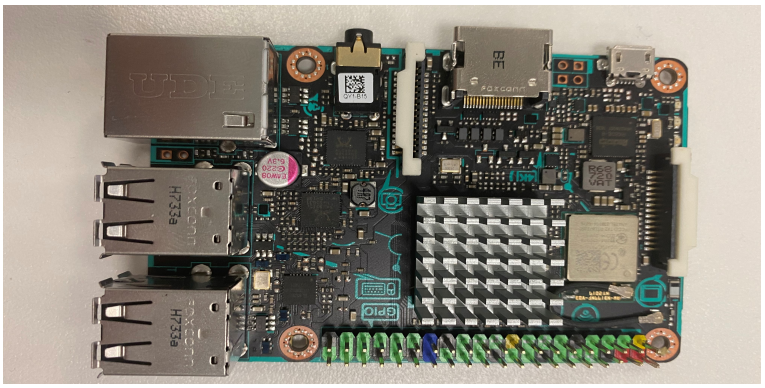


Figure 3: image of Tinker Board

3.2 Localization

Localization is all about how to know the exact spatial coordinates of each robots. We propose following 3 options, but we'd rather take this first option as it's sophisticated and more suitable for a quick test. Here are the possible options:

1. Option 1: Markers in STAR Labs'tracking system
2. Option 2: Camera on robot - april tag on ceiling
3. Option 3: Camera on ceiling - down looking - april tag on robot

Through real tests, we found the tracking system is suitable to complete such kind of works,so we use the Optitrack as our localization system.fig.6 shows the Optitrack multi-camera system and fig.1(b) shows the visulization window.

3.3 Control algorithm

Through reading a lot of papers, we decided to take the approach called Non-holonomic optimal reciprocal collision avoidance (NH-ORCA).

This approach builds on Optimal Reciprocal Collision Avoidance (ORCA) and extends it toward non-holonomic reciprocal collision avoidance. The robots are controlled to stay within a maximum tracking error ϵ of an ideal holonomic trajectory. Control inputs for optimal tracking are derived from mapping holonomic onto non-holonomic velocities.

Figure 4: our localization system: Optitrack

Althoough there exists another simpler version [Sna+10], which is exactly what we implemented, however we still try to explain this detailed version.

3.3.1 ORCA

ORCA is a velocity-based approach to collision avoidance that provides a sufficient condition for guaranteeing collision-free motion among multiple holonomic robots. Given a group of n disk-shaped robots with radius r_i and velocity $\mathbf{v}_i \in \mathbb{R}^2$ at position $\mathbf{p}_i$ in the plane $\mathbb{R}^2$, each robot tries to reach an assigned goal point $\mathbf{g}_i$ by selecting a preferred velocity $\mathbf{v}_i^{\text{pref}} \in \mathbb{R}^2$. The objective is to choose an optimal $\mathbf{v}_i$ which lies as close as possible to $\mathbf{v}_i^{\text{pref}}$, such that collisions among the robots are avoided for at least a time horizon τ.

In the case of holonomic robots with velocities $\mathbf{v}_H \in \mathbb{R}^2$, the velocity obstacle for robot $i \in [1, n] \subset \mathbb{N}$ with r_i at $\mathbf{p}_i$ induced by any robot $j \in [1, n], j \neq i$, with r_j at $\mathbf{p}_j$ is defined as the set of relative velocities $\overline{\mathbf{v}} = \mathbf{v}_{H_i} - \mathbf{v}_{H_j}$ between robots i and j

$$VO_{ij}^{\tau} = \{\overline{\mathbf{v}} \mid \exists t \in [0, \tau], t \cdot \overline{\mathbf{v}} \in D\left(\mathbf{p}_j - \mathbf{p}_i, r_i + r_j\right)\}$$

with $D(\mathbf{p}, r) = \{\mathbf{q} \mid \|\mathbf{q} - \mathbf{p}\| < r\}$ the open ball of radius r. The set of collision-free velocities $ORCA_{ij}^{\tau}$ for robot i with respect to robot j can geometrically be constructed from $VO_{i|j}^{\tau}$ (see Fig. 2 left and center). First, the minimum change

$$\mathbf{u} = \left(\underset{\mathbf{v} \in \partial VO_{ij}^{\tau}}{\operatorname{argmin}} \left\|\overline{\mathbf{v}} - \left(\mathbf{v}_i^{opt} - \mathbf{v}_j^{opt}\right)\right\|\right) - \left(\mathbf{v}_i^{opt} - \mathbf{v}_j^{opt}\right)$$

which needs to be added to $\bar{v}$ to avoid a collision, is computed. v_i^{opt} is the optimization velocity, set to the current velocity $v_{H_i}^{\text{vurent}}$ of the robot. This gives good results as shown in [2]. Then $ORCA_{i|j}^{\tau} = \left\{\mathbf{v}_{H_i} \mid \left(\mathbf{v}_{H_i} - \left(\mathbf{v}_i^{opt} + c\mathbf{u}\right)\right) \cdot \mathbf{n} \geq 0\right\}$ follows as described in [2]. n denotes the outward normal of the boundary of VO_{ij}^{τ} at $\left(\mathbf{v}_i^{opt} - \mathbf{v}_j^{opt}\right) + \mathbf{u}$, and c defines how much each robot gets involved in avoiding a collision. $c = \frac{1}{2}$ means both robots i and j help to equal amounts to avoid colliding with each other; $c = 1$ means robot i fully avoids collisions with a dynamic obstacle j. Likewise, the velocity obstacle can be computed for static obstacles following [2]. The set of collision-free velocities for robot $i, ORCA_i^{\tau}$, is given by

$$ORCA_i^\tau = S_{AHV_i} \cap \bigcap_{j \neq i} ORCA_{i|j}^\tau$$

with S_{AHV_i} the set of allowed holonomic velocities under the kinematic constraints of robot i. For holonomic robots, $S_{AHV_i} = D\left(0, V_{H_i}^{\max}\right)$. The optimal holonomic velocity for robot i is to be found as

$$\mathbf{v}_{H_i}^* = \underset{\mathbf{v}_{H_i} \in ORCA_i^\tau\}}{\operatorname{argmin}} \left\| \mathbf{v}_{H_i} - \mathbf{v}_i^{pref} \right\|$$

3.3.2 Selection of non-holonomic controls

In this section, the control inputs (v, ω) for optimal tracking of a given holonomic velocity $\mathbf{v}_H$ are found. The controls for the non-holonomic robot are chosen as those that minimize the tracking error ε_H, while achieving the correct orientation in the fixed given time T. If this is impossible due to the robot's constraints, the robot performs a turn in place by rotating at maximum speed until the correct orientation is reached, i.e. $\omega = \min\left(\frac{\theta_H}{T}, \omega_{\max}\right)$. In general, t_1, θ_H and ω are related by $\omega = \frac{\theta_H}{t_1}$ With everything else fixed, the linear velocity that minimizes Eq. (2) is given by

$$v^* = \frac{V_H t_1 \sin(\theta_H)\,\omega}{2\left(1 - \cos(\theta_H)\right)} = V_H \frac{\theta_H \sin(\theta_H)}{2\left(1 - \cos(\theta_H)\right)}$$

The optimal linear velocity might not be feasible due to the limits on the linear and angular velocities. Therefore, the optimal controls are

$$\begin{aligned} &R_{A1} : \omega = \tfrac{\theta_H}{T} \leq \omega_{\max} \text{ and } v = v^* \leq v_{\max,\omega} \\ &R_{A2} : \omega = \tfrac{\theta_H}{T} \leq \omega_{\max} \text{ and } v = v_{\max,\omega} \\ &R_B : \omega = \omega_{\max} \text{ and } v = 0 \end{aligned}$$

If the optimal controls are chosen, the maximum tracking error $\varepsilon_H^2(\mathbf{v}_H)$ committed in each of the regions are derived from Eq. (2) and Eq. (8) and given by

$$\begin{aligned} &R_{A1} : \varepsilon_H^2 = \left(\tfrac{2(1-\cos(\theta_H)) - \sin^2(\theta_H)}{2(1-\cos(\theta_H)}\right) T^2 V_H^2 \\ &R_{A2} : \varepsilon_H^2 = V_H^2 T^2 - \tfrac{2V_H T^2 \sin(\theta_H)}{\theta_H} v_{\max,\omega} + \tfrac{2T^2(1-\cos(\theta_H))}{\theta_H^2} v_{\max,\omega}^2 \\ &R_B : \varepsilon_H = V_H t_1 = V_H \tfrac{\theta_H}{\omega_{\max}} \end{aligned}$$

3.4 Code

Currently, we wrote two subscripts based on the open-source code RVO2.

First, we transfer it to python interface for faster iterations.

Then, we re-implement the sample simulation code which shall be discussed in following sections.

150 lines (150 sloc) 3.94 KB Raw Blame

In [1]:
```
import pyrvo2
import numpy as np
import math
```

In [2]:
```
from pyrvo2 import *
```

In [3]:
```
goals=[];
def  setupScenario(sim):
    sim.setTimeStep(0.25);
    sim.setAgentDefaults(15.0,  10,  10.0,  10.0,  1.5,  2.0,np.array([0,0]));
    for i in range(250):
        sim.addAgent(200.0 *np.array([math.cos(i * 2.0 * math.pi / 250.0),
                                      math.sin(i * 2.0 *  math.pi / 250.0)]))
        goals.append(-sim.getAgentPosition(i));

def updateVisualization(sim):
    print(sim.getGlobalTime());
    for i in range(sim.getNumAgents()):
        print(sim.getAgentPosition())

def setPreferredVelocities( sim):
    for i in range(sim.getNumAgents()):
        goalVector = goals[i] - sim.getAgentPosition(i);
        if (np.linalg.norm(goalVector) > 1.0):
            goalVector = goalVector/np.linalg.norm(goalVector)
        sim.setAgentPrefVelocity(i,  goalVector);

def reachedGoal(sim):
    for i in range(sim.getNumAgents()):
        if (np.linalg.norm(sim.getAgentPosition(i) - goals[i]) > sim.getAgentRadius(i)):
            return False;
    return True;
```

Figure 5: Sample code of ORCA,part 1

After that, we wrote the messaging code for the bot and tries to combine that with real robots, but we have still not tested yet.

In [4]:
```
import pygame as pg
screen_size=[400,400]
world_size =[500,500]
screen=pg.display.set_mode(screen_size)
screen_size= np.array(screen_size)
screen_center = screen_size/2
world_center = np.array([0,0])
world_size = np.array(world_size)
def getScreenPos(world_pos):
    return (world_pos-world_center)/world_size*screen_size+screen_center
def getScreenRadius(radius_world):
    return radius_world/np.max(world_size)*np.max(screen_size)

def visualize(sim):
    screen.fill(pg.Color("white"))
    for i in range(sim.getNumAgents()):
        pos = getScreenPos(sim.getAgentPosition(i))
        radius = getScreenRadius(sim.getAgentRadius(i))
        pg.draw.circle(screen,pg.Color("blue"),(pos[0],pos[1]),radius,0)
    pg.display.update()
```

```
pygame 2.0.0 (SDL 2.0.12, python 3.8.3)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

In [5]:
```
import time
start_time = time.time()
sim = RVOSimulator();
setupScenario(sim);
while (not reachedGoal(sim)):
    setPreferredVelocities(sim);
    sim.step();
    visualize(sim)
end_time = time.time()
print(end_time-start_time)
```

```
76.2096107006073
```

Figure 6: Sample code of ORCA,part 2

4 System Evaluation

Simulation and real robot test,adding some data analysis and image.

4.1 Simulation Test

On simulation with the code in figure3, we test a group of 250 circle robots, each have the same radius and other parameters, except their goal positions.

The robots are placed along a circle and they try to reach the opposite point along the diameter of the circle. The result of figure4 shows the process of ORCA moving as the time goes by. First the

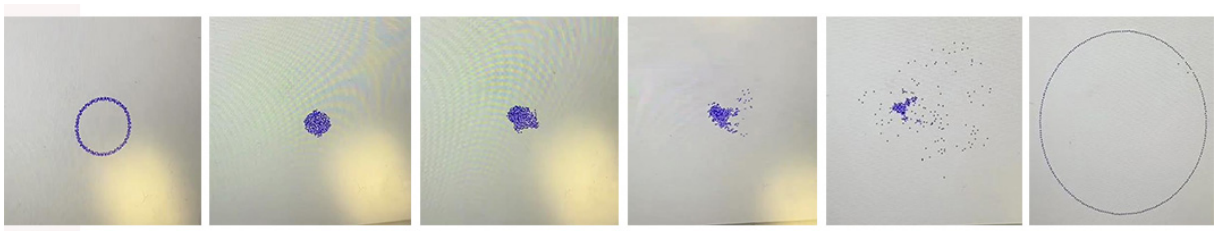


Figure 7: simulation results

robots are moving and form a cluster. Then with the help of algorithm, the robots success to find their way out while avoiding collsions. Finally, they suceeded in reaching their goals respectively.

4.2 Real-robot Test

We have evaluated the proposed collision avoidance method and the theoretical results by experiments with real robots.

A group of four turtle bots robots is used in the experiments.

To tracing these turtle bots,we use the OptiTrack Tracking system with markers on the turtlebots.

After setting up a single robot's handware and softwares, we test the remote control of the turtlebot through teleop. And it successfully run as expected, which means the command node cmd_vel works effectively.

Then we build new stystems for each robots by copying SD cards.

After that,we setuped the network connectiosn by export hostname and master_uri. Make use the tracking system,all robots and a additional master laptop share in the same wifi environment.

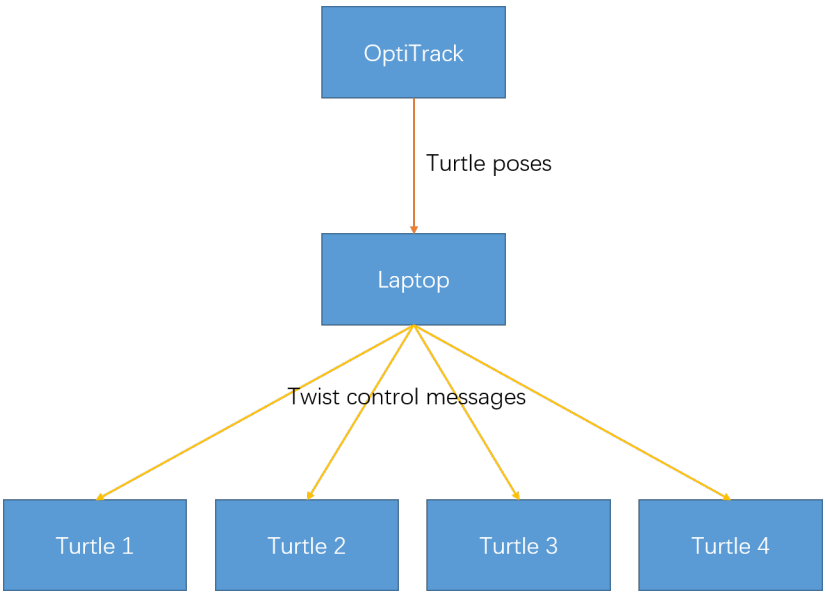


Figure 8: System Diagram

To evaluate the system , we would like to started two test:

4.2.1 Two turtle bots

Two robots A and B will try to exchange their positions.In figure a, they started to move.In figure b, they found they will collide so they changed their velocities. Then in figure c, they succeeded without any collisions.

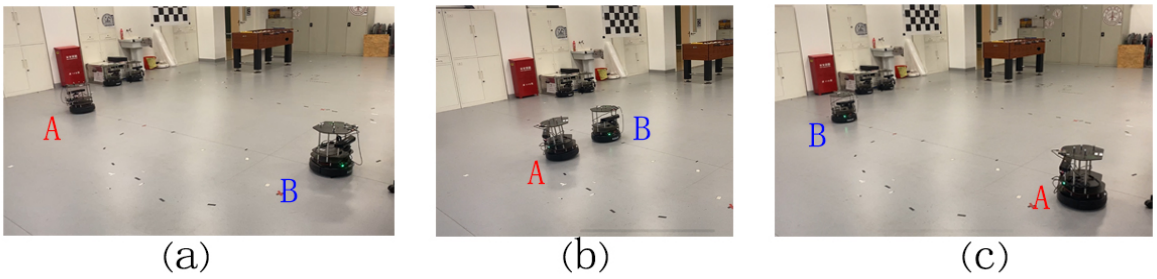


Figure 9: Two real turtle bots results

4.2.2 Four turtle bots

Four robots A,B,C,D will try to exchange their positions.In figure a, they started to move.In figure b, they found they will collide so they changed their velocities. Then in figure c, they succeeded without any collisions.

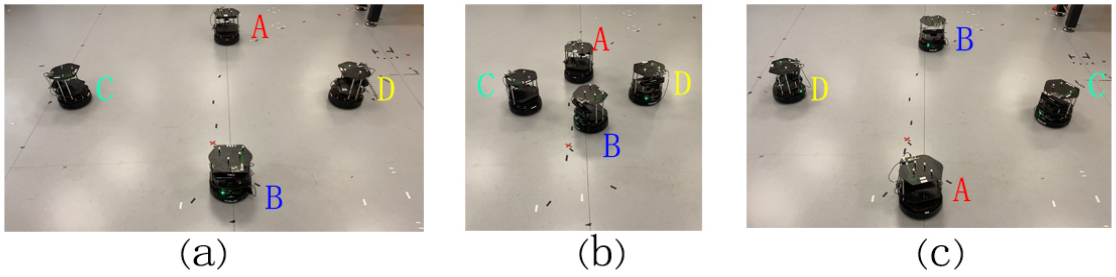


Figure 10: four real turtle bots results

Here we plot their trajectories:

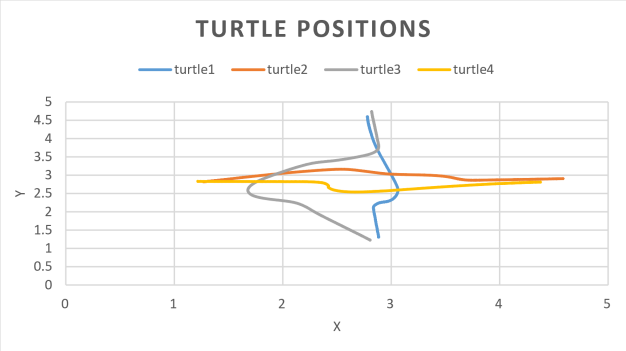
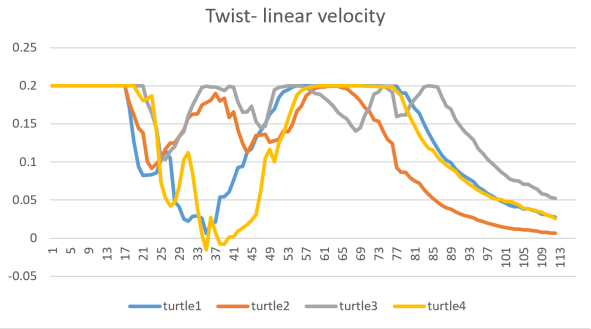
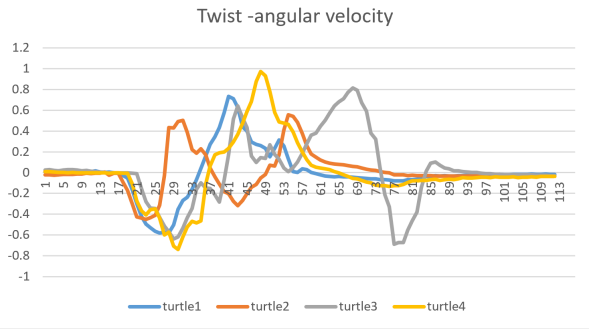


Figure 11: robot trajectories

And also their control commands along time:





5 How-to

5.1 Enable network connections

First, create master in your laptop:

```
$ roscore
```

Then you have to bringup all robots with their namespaces after sshing into their system:

```
$ ROS_NAMESPACE=turtleX roslaunch turtlebot_bringup minimal.launch
```

Make sure the master uri (laptop ip) and hostname(robot ip) is correct

5.2 Setup Localizations

1. Open the tracking system 'multi_turtle.ttp' on the tracking system's laptop (not the master laptop) under the same wifi.
2. Move each robot into the tracking scene. Make sure their orientation is x-axis positive.
3. Then create rigidbodies from their markers, and change their name to turtleX where X is the id labelled on the robot (1 to 4).
4. Now, on your master laptop, start your vrpn_client_node. You shall find 4 rigidbodies, turtle1,turtl2,turtle3,turtle4.

   With rostopic echo, you shall find all 4 turtle topics like

```
    \turtleX\odom
    \vrpn_client_node\turtleX\pose
    ..
```

5.3 Control Codes(including algorithms)

Here you can git clone the codes. Then you shall build as instructed in Readme.md.

After all built and setup completed,you shall cd into the python folder.

Note that you should set turtle positions in the turtleCtrl.py as you observe on your tracking system.

Then run:

```
python turtleCtrl.py
```

Then four robots shall start moving.

6 Conclusion

To sum up, our project aims at setting up a multi-robot coordination system and we choose the turtle 2 robot as the main vehicle.

To accomplish this, we did following tasks:

1. Setup robot for communication and control
2. Localization of the robots in the lab area
3. Implemented and NH-ORCA algorithms on the robots

We equipped the tinkerboards on the robots and localized them with the labs' tracking system.

Besides, we transfer the ORCA algorithm to a python interface(running c in backend with openmp) and then extend it to a Non-holonomic version on python and send twist control commands through rospy.

The plan had been made and the overall process has been proved to be promising.

6.1 Future work

As discussed in the paper, guaranteeing collision-free motion among multiple holonomic robot deserves more analysis and study, and how to propose a velocity-based approach to collision avoidance is an interesting topic to explore. Additionally, it would be interesting to extend the method here presented to other non-holonomic vehicle dynamics. We believe this can be achieved by modifying the set of allowed holonomic velocities S_{AHV} .Eventually, the method could be generalized for higher dimension and applied to underwater or aerial robots.

References

[CFK97] Y Uny Cao, Alex S Fukunaga, and Andrew Kahng. "Cooperative mobile robotics: Antecedents and directions". In: *Autonomous robots* 4.1 (1997), pages 7–27.

[FIN04] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. "Multirobot systems: a classification focused on coordination". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34.5 (2004), pages 2015–2028.

[Fox+00] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. "A probabilistic approach to collaborative multi-robot localization". In: *Autonomous robots* 8.3 (2000), pages 325–344.

[INS00] Luca Iocchi, Daniele Nardi, and Massimiliano Salerno. "Reactivity and deliberation: a survey on multi-robot systems". In: *Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*. Springer. 2000, pages 9–32.

[Jia+17] Y. Jiang, M. Zanon, R. Hult, and B. Houska. "Distributed algorithm for optimal vehicle coordination at traffic intersections". In: *In Proceedings of the 20th IFAC World Congress, Toulouse, France*. 2017, pages 12082–12087.

[KNT92] S. Kato, S. Nishiyama, and J. Takeno. "Coordinating Mobile Robots By Applying Traffic Rules". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Volume 3. 1992, pages 1535–1541.

[Sna+10] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha. "Smooth and collision-free navigation for multiple robots under differential-drive constraints". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pages 4584–4589. `https://doi.org/10.1109/IROS.2010.5652073`.

[TRS00] Eduardo Todt, Gustavo Rausch, and Raúl Suárez. "Analysis and classification of multiple robot coordination methods". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Volume 4. IEEE. 2000, pages 3158–3163.

[YJC13] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. "A Survey and Analysis of Multi-Robot Coordination". In: *International Journal of Advanced Robotic Systems* 10.12 (2013), page 399. `https://doi.org/10.5772/57313`. eprint: `https://doi.org/10.5772/57313`. `https://doi.org/10.5772/57313`.