COOPERATIVE MANIPULATORS

By

MOHAMAD FIRDAUS BIN RAZALI

15267

Final Year Project Report submitted as partial

fulfillment of the requirement of

Final Year Project

SEPTEMBER 2013

CERTIFICATION OF APPROVAL

Cooperative Manipulators

by

Mohamad Firdaus Bin Razali

A project dissertation submitted to the

Mechanical Engineering Program

Universiti Teknologi PETRONAS

in partial fulfillment of the requirement for the

BACHELOR OF ENGINEERING (Hons)

(MECHANICAL ENGINEERING)

Approved by,

____

(Prof Dr Thirumalaiswamy Nagarajan)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

September 2013

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

____

MOHAMAD FIRDAUS BIN RAZALI

ACKNOWLEDGEMENT

First and foremost, the author would like to express his highest gratitude to his supervisor, Dr Thirumalaiswamy Nagarajan for all the motivation, knowledge and valuable advices that he has given. The guidance and supervision from him have granted his many opportunities to explore, learn and experience throughout the project.

The author is also very grateful to Dr Tadimalla Varaha Venkata Lakshmi Narasimha Rao who had provided the author constructive advices and valuable assistances in accomplishing the project.

This project will not be successful without helps from Mr Ashwin, Mr Fudzail and Ms Tay Mei Chen for aiding the author completing this experimental study. Their kindness and hospitality provide the author a comfortable and conducive working environment.

Finally, the author would like to extend his appreciation to Universiti Teknologi PETRONAS for providing the facilities and many learning opportunities, and special thanks to his family and friends who constantly support him throughout the project. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The objective of this project is to achieve the cooperation between two mobile robots. These two robot will be programmed to lift a certain object or load and carrying the load simultaneously. A general programming controller will be use in this project to achieve the main objective to study how the mobile robot work. This project will present the experimental findings to create cooperative task between mobile robot through a single command controller and the manipulation of control between them. By doing this project, cooperative manipulation between any controllable robots to do the same exact movement can be done. This is important on how we could manipulate the robot doing task or activity following the controller desirable outcome. The project mainly focusing on programming task, as the mobile robot are already available. The study of command control is very important to achieve the project objective.

Chapter 1

Introduction

1.1 Project Background

The adoption of cooperative manipulators is often required to execute wide type of task and movement. Among the important task implemented to these robots are carrying large or heavy load and mating the mechanical parts. When the cooperative multi arm system is employed for the manipulation of a common object, it is very important to control both the absolute motion and movement of the object. To achieve this complicated task, the programming mapping between force and velocities are the main criteria to be manipulated.

This project will study on formation control for coordinating multiple mobile robots using the leader-following formation approach method. Specified command and task will be given to these robots and data collected will be analyzed. One or more robots will act as a leader while the other will be the follower navigating the set environments in leader-following formation. Figure below show the cooperative task between two arm robots to lift a load.

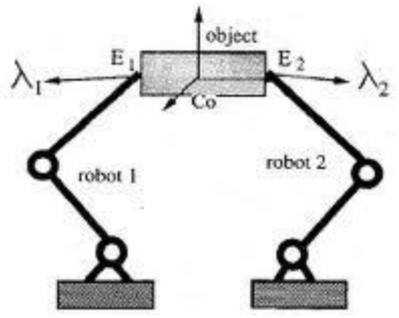


Figure 1: Robot arm lift an object.

1.2 Problem Statement

Controlling two different robots to do the same task is highly difficult. Many problems will be face including formation, navigation and coordination of motion. Thus, this project will study the robot controlling program to allow to different robot lifting a load and move around through navigation.

1.3 Objective

This project goal is to experiment on how two different mobile robot to carry a single load simultaneously and move by navigation. In the process, a lightweight load will be chosen and will be put between robots. While the robot moving from one point to another, the load will be expected to stay above the robot. The cooperation between two robots transferring the load from one point to another will definitely approve the theory of cooperative manipulators.

1.4 Scope of Study

The study will be conducted by experiments using mobile robots. Specified command and task will be given to the three robots based on the leader-following formation approach method and observation and data collection will be made during the experiment.

The experiment will be programmed in the structural programming or C++ and the robot command of movement will be build. The mobile robot, AmigoBots will be used and assigned will simple task to make some movement or trajectory together at almost simultaneously to carry a lightweight load. The load will be moved or transfer by the robots from one point to another to prove the theory of cooperative manipulation between the robots. The coordinate data or result will be collected after the experiment to show the targeted trajectory successful or not and fault tolerance will be calculated.

This experimental study being done to prove that cooperation between the robots to carry the load without had the load fall down while doing trajectory movement.

Chapter 2

Literature Review

2.1 Multiple Mobile Robots

In recent decade, mobile robots application has expanded to various areas which include underwater, space exploration and hazardous environment. The demand for mobile robots to perform more complex tasks is increasing and with that, coordination between multiple mobile robots will allow such task to be completed. A group of robots can provide data redundancy and contribute cooperatively to perform given task with greater reliability, speed and cost reduction compared to a single mobile robots (Arai, T., et al., 2002).

To coordinating multiple mobile robots, formation control is required. Formation control refers to the ability to control the relative position and orientation of robots in a group, while allowing them to move as a whole as decribed by Kuppan Chetty R.M., Singaperumal, M. and Nagarajan, T., (2011). Kuo-Yang Tu and Min-Tzung Chang Chien (n.d.) addedd that formation control is also how multiple mobile robots maintain formation to overcome unexpected events in crucial situation to finish task given.

2.2 Pattern Formation

According to Bahceci, E., Soysal O. and Sahin E., (2003), pattern formation refers to the coordination of a group of mobile robot to get into and maintain formation with a certain shape. There two groups of pattern formations which are centralized pattern formation and distributed pattern formation.

In centralized pattern formation method, the group motion will be planned by a single computational unit as it oversees the others as stated by Belta C. and Kumar V., (2002). Each robot will then receive their respective movement via communication network. Centralized patter formation depends on a single central unit that oversee the entire group with communication channel between central unit and other robots as the receiver which makes it more costly, less robust to failures, and less scalable to the control of large number of robots.

Decentralized pattern formation uses local communication and sensor for each robots and it tends to be more scalable, robust and easier to build but limited in variety and precision of formations according to Balch T. and Hybinette M (2000) and Dudenhoeffer D.D. and Jones M.P., (2000). This pattern applies on multi robot system. Multi-robot system are preferred compared over a single central robot in centralized pattern due to the robustness and it can be improved by incorporating adaptation mechanisms that responds to change in environment and individual robot capabilities.

2.3 Formation Control Methods

According to Gomman J., et al, (2009), there are three main methods to formation control for multiple robot each with their own advantage and disadvantage. The three main methods are behavioral, virtual structure and leader-following.

Behavioral approach is define as a set of desired behaviors for each individual in the group, and measures them such that desirable group behavior emerges without explicit model of the subsystem or the environment. Balch T. and Arkin R.C. (1998), state that in multi robot system, behavioral approach is described as implementation of others navigational behaviors on the formation to derive control strategies for goal seeking, collision avoidance and formation maintenance. The advantage for this approach is that it is natural to derive control strategies when vehicles have multiple objectives, and a clear feedback is included through communication between group members. The disadvantages are that the group behavior cannot be explicitly defined and it is difficult to analyze the approach mathematically and guarantee the group stability according to Gomman J., et al, (2009).

For virtual structure approach, the whole formation is described as a single, virtual, structure and it acts as a unit. The governing control for each individual robots is derived by defining the motion of the virtual structure and interpreted it into the desired movement for each mobile robots as stated by Do, K.D (2007). Beard R., Lawton, J.

and Hadaegh, (1999), mentioned that virtual structure has been achieved by having all members of the formation tracking assigned nodes which move into required configuration. Advantage of the virtual structure approach is that is easy to follow the coordinated behavior for the group and information can be maintained very well during maneuvers. The disadvantage is that the formation has to maintain the exact same virtual structure all the time limiting the application's potential.

In leader-follower formation approach, some of the robot act as leaders in the group while the others as followers following the leaders' movement as described by Wang P.K.C., (1992). Kuppan Chetty, R.M., et al., (2011), added that the main objective of the robot acting as the follower is to position themselves relative to the leader and maintain a required distance and orientation among each others. Complex formation is possible to achieve by controlling relative positions and orientation of the leader and followers. Advantages of this approach are that it is easy to understand and implement and the formation can still be maintained even if the leader is disturbed. The disadvantages is that there no explicit feedback from followers to the leader especially in dynamically changing, unknown, unstructured environments.

2.4 Layers Architecture for Leader Follower Formation

Fig.1 shows the hybrid formation control strategy consisting of layered behavior based architecture. The lower level consists of navigation control for robots and the supervisor level is a decentralized leader-follower formation controller. Formation and navigation switch is done based on the local information and the role of robots in the group.

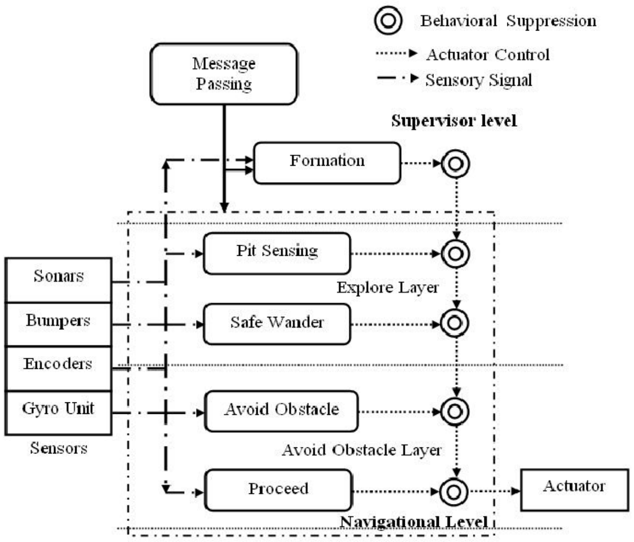


Figure 2: Layered Formation Control Architecture

Navigation capabilities of the robots is achieved at lower level which consist of two layers which are the explore layer and avoid obstacle layer. The higher level formation control consists of supervisor layer required for higher level operation such as formation and communication.

In leader-follower formation approach, the robot acting as leader of the group navigate the environment with the lower level behavioral architecture. At this point, the functionality of the system is divided into simpler task or behaviors that are manipulated sequentially and transmit its relative position and orientations through higher level behavior of message passing. The followers execute the formation while the leader navigates independently.

Detail of some of the behaviors according to Kuppan Chetty R.M, et al, (2011) is as follows:

a) Heading: This behavior process provides approximate heading values for the safe wandering and obstacle avoidance behaviors by processing the positioning data, providing the robot current position and orientation at anytime ion two dimensional workspace.

b) Avoid Obstacle: By using the sensors, this behavior allows the robots to avoid obstacle without colliding and to navigate through the environment. This behavior is initiated when the robot sensor detects an obstacle and it will manipulate the wheel’s translational and rotational velocity.

c) Safe-wandering: With piece wise constant velocity, this behavior guide the robot through the environment by turning left or right at regular intervals at set angle. This allows the robot to explore the environment thoroughly and look for the goal.

d) Message passing: According to Hu H., et al (1998) and James L.C. and Patrick R., (1993), message passing behavior allows the robots to exchange information such as position, orientation and velocity by using the explicit socket communication capability through wireless links. In leader-following formation approach, this behavior provides the necessary interaction between the leader and follower by allowing the leader to current tasks or behavior to the follower and the follower will navigate through the environment according to the leader’s information and also allows leader to know the follower’s current position.

e) Formation: This behavior manipulates the required wheel velocities of the follower when the leader path is known to maintain the follower position relative to the leader with specified separation and orientation.

The proposed leader-follower formation control approach was experimented using two Pioneer P3DX mobile robots research platform and it was concluded that the dynamic switching between the behaviors and robot helps the follower to trade their roles with the leader to avoid obstacle in their path while maintaining the desired formation.

Chapter 3

3.1 Methodology

The project methodology as the shown below in the Figure 3.

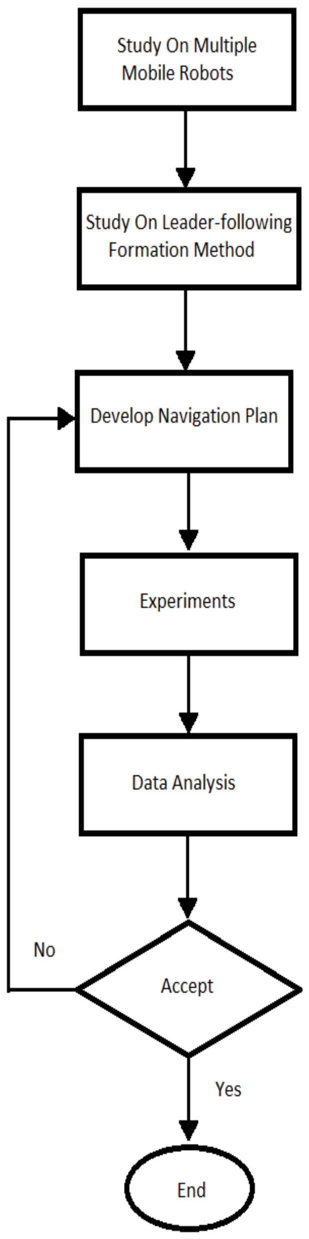

Study On Multiple Mobile Robots
Study On Leader-following Formation Method
Develop Navigation Plan
Experiments
Data Analysis
No
Accept
Yes
End

Figure 3 : Project Methodology

At the early stage of the project, study will be done on the multiple mobile robots coordination and formation. The research is conducted to acquire better understanding on the subject through literature reviews, journals reading, and internet research. Robot-to-PC connection will be established with client-server relationship. The next step is to connect the three AmigoBot mobile robots using the wireless network. After forming connection between the robots, testing and trial run is done to test the working principle and maneuverability. Trajectory plan will be developed and uploaded to the robots. Experiment will be done using various trajectory plans and set environments. The expected result from this experimental study is the robots will maintain formation, with almost perfect navigation and orientation less than 10% fault tolerance.

3.2 Gantt Chart and Project Milestone

No	Detail/Week	FYP 1 (20 May 2013 – 23 August 2013)														FYP 2 (23 September 2013 – 27 December 2013)													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1.	Selection of Project Title	🟦	🟦																										
2.	Preliminary Research Work			🟦	🟦	🟦																							
3.	Submission of Extended Proposal						🟦																						
4.	Proposal Defense							🟦																					
5.	Establish Robot-to-PC connection							🟦	🟦	🟦	🟦																		
6.	Establish Wireless Communication Network										🟦	🟦	🟦																
7.	Testing and Trial run										🟦	🟦	🟦	🟦	🟦														
8.	Develop Trajectory Plan														🟦														
9.	Submission of Interim Draft Report													🟦															
10.	Submission of Interim Report														🟦														
11.	Develop trajectory plan															🟦	🟦	🟦											
12.	Experiments and data collection																		🟦	🟦	🟦	🟦							
13.	Submission of Progress Report																						🟦						
14.	Experiments and data collection																						🟦	🟦	🟦	🟦	🟦		
15.	Pre- SEDEX																									🟦			
16.	Submission of Draft Report																										🟦		
17.	Submission of Dissertation (soft bound)																											🟦	
18.	Submission of Technical Paper																											🟦	
19.	Oral Presentation																											🟦	
20.	Submission of Project Dissertation (Hard Bound)																												🟦

3.3 Tools and Requirement

This project will be using software and hardware as tools to achieve the cooperation of movement and coordination among robots. The software use in this project is ARIA and MobileEyes. ARIA use to input the trajectory plan while the MobileEyes use to monitor robot position, linear velocities and angular velocities.

The hardware of this project is the AmigoBot Mobile Robot. The mobile robots will be used to trajectory the cooperative movement.

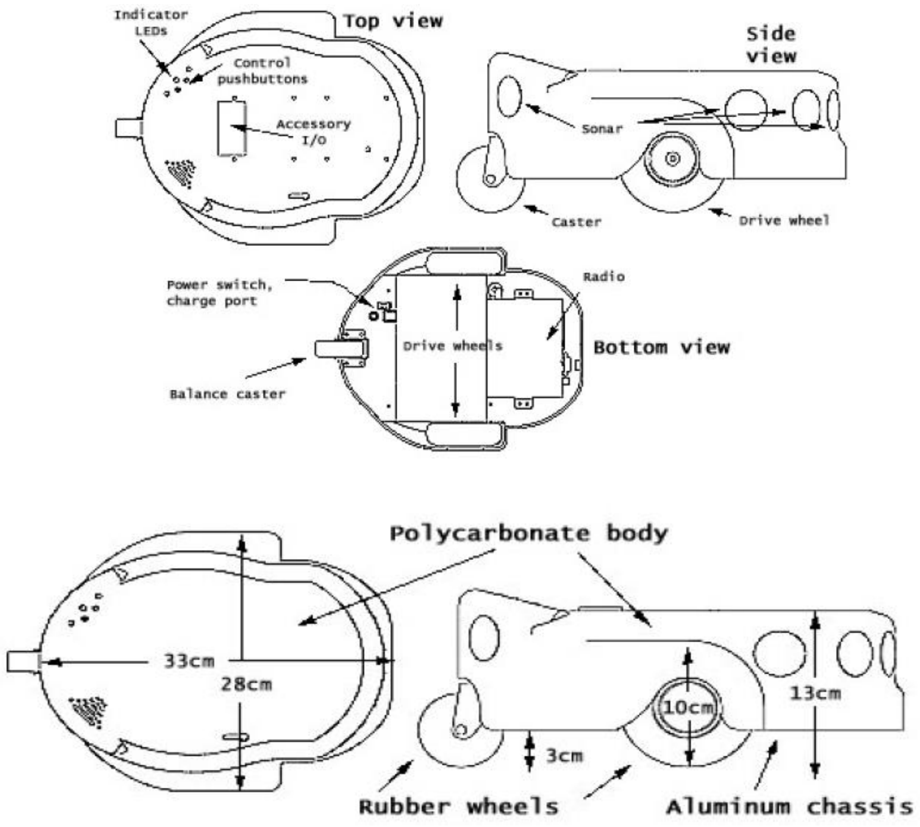


Figure 4: The AmigoBot

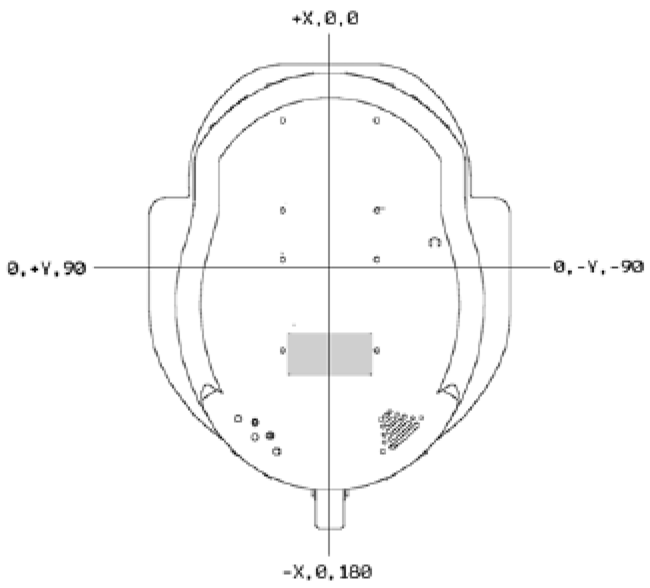


Figure 5: The AmigoBot axis direction

Robot specification:

i. Weight: 3.6 kg

ii. High maneuverability: 750 mm/sec translation speed, 300 degree/second rotation, turns in place

iii. High-impact resistant polycarbonate body and lightweight aluminum chassis

iv. Driven by 12 VDC motor

v. Eight range finding SONAR, 360 degree coverage

3.4 Software Required

Software involved includes Microsoft Visual Studio Express 2010, Python Server 2.7 MobileSim and advance robot interface for application (ARIA) library. Microsoft Visual Studio Express 2010 software is used to develop and compile control program by using action group available in ARIA library to dynamically control robot's velocity, heading, relative heading, and other motion parameters either through simple low-level commands or through its high-level Actions infrastructure. Python Server 2.7 allows communication between leader and follower robots by navigating information transfer via wireless network. MobileSim software is for simulating mobile robots platforms and their environments and for debugging and experimentation of the compiled command structure.

3.5 Experiment Description

The experiment that will be done by the mobile robots will be conducted by using two mobile robots to test the coordination of movement to achieve the target of cooperation between them. Thus, these robots will be tested to carry a light load together and performing some movement along. One robot assigned ‘leader mode’ while the other two robots is in ‘follower mode’. Each robot will have individual onboard PC connected to the robot using a 9 pin serial-to-USB tether cable and each PC will be connected to each other via a wireless communication network using wireless router.

For the first experiment, two robots will carrying a load together and moving in a straight line. The robots will be moving in the straight line for 4000 millimeters or 4 meters in a straight line at almost simultaneously. The coordinate for the first experiment is 10000 for the x-axis and 0 for y-axis.

The targeted trajectory for both robots are as shown below;

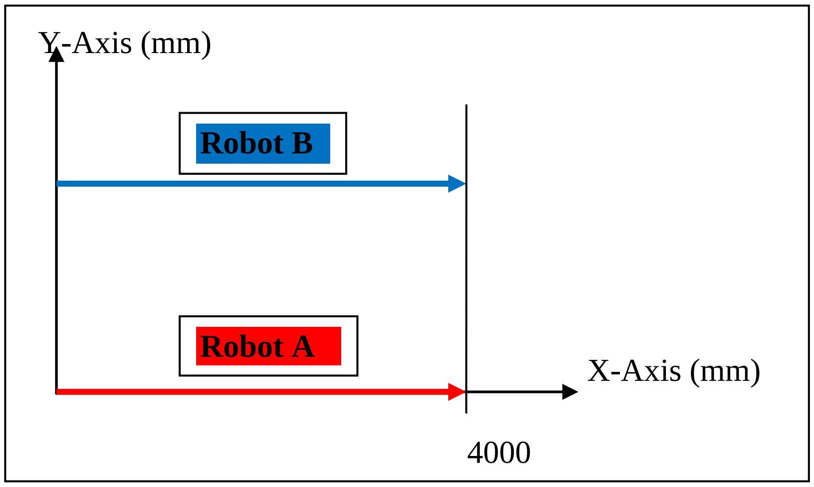


Figure 6: Experiment 1

As these robots using the ‘leader and follower’ coordination format, some of these robots will be assigned as leader and one as follower. The gap between the leader and follower reaction is approximately 100 milliseconds thus the far trajectory movement, the bigger fault tolerance could happen. Fault tolerance is happen when the robots are departing from each other from doing the same movement simultaneously. The fault tolerance targeted for this experiment is less than 10 percent.

As for the second experiment, these robots will be assign to do more complex trajectory that is the zigzag movement. Both robots will carry the same load to move in zigzag trajectory together. The coordinates for these robots to follow is (1500, 1500), (3000, 1500), (4500, 1500) and (6000, 1500).

The targeted trajectory for the second experiment are as the figure shown below;

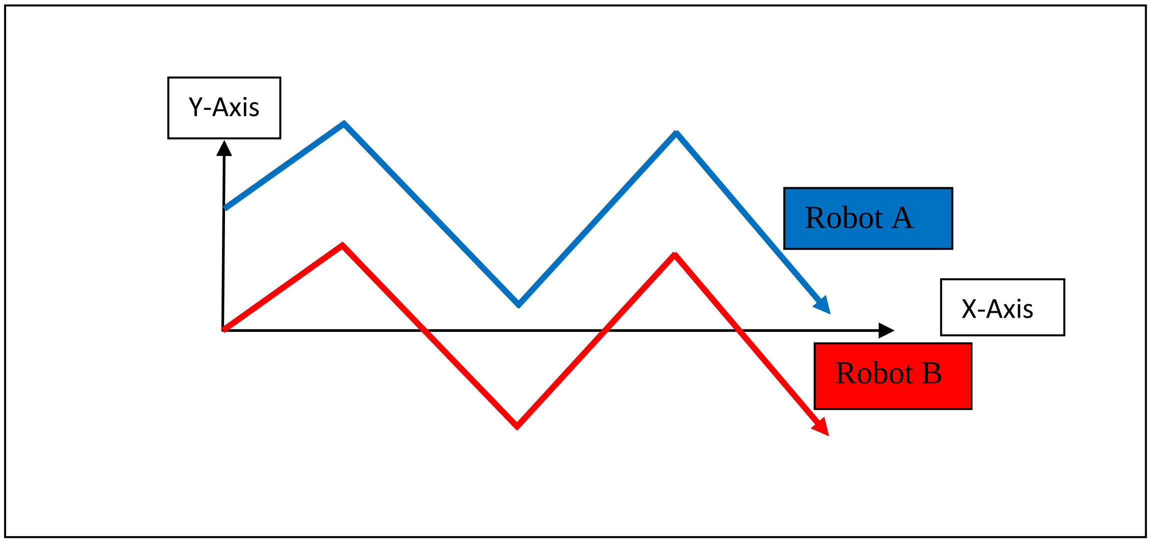


Figure 7: Experiment 2

In these two experiments, the data on the spatial distance between the robots is recorded. Error calculation will be done by comparing starting and end spatial distance using the robots relative coordinates. Modification will be made if the fault tolerance recorded is more than 10% of targeted trajectory.

Chapter 4

Result and Discussion

4.1 Control Program

A control program is design to fulfill experiment requirements include performing given task and collecting required data. The control program is developed using Microsoft Visual C++ express utilizing ARIA library. ARIA library contain various action groups and among them are ones that allow communication between robots and onboard PC and controlling the robot’s motor and sensors. Following are the main function for the designed control program:

- Build robot-to-PC and PC-to-server connection
- Assign ‘leader mode’ and ‘follower mode’ to respective robots
- Goal setting by defining required X and Y coordinates
- Transmit and receive coordinate information and differentiate between leader and follower coordinates
- Behavior control which include stall recovery, pit sensing and obstacle detection and avoidance
- Trajectory plan for more complex trajectory path

The program utilizes Python Server to allow message passing from the leader to the followers. Server receive current position information from all the robots in real time at a rate of 100 milliseconds per cycle and select only the coordinates from the leader and transmit it to follower’s onboard PC and updates the follower current goal. This process repeats itself until the leader reaches the target. The server can be installed in either one leader or follower PC.

4.2 Main Control Structures

For cooperative manipulators that have been implied in this experiment, one robot will be assigned as leader and one other as the follower. This mean only one laptop or tablet above the robot will be running the C++ programming code while the other one is none. Both of the robots will be connected to the Python Server and the programming will be conducted.

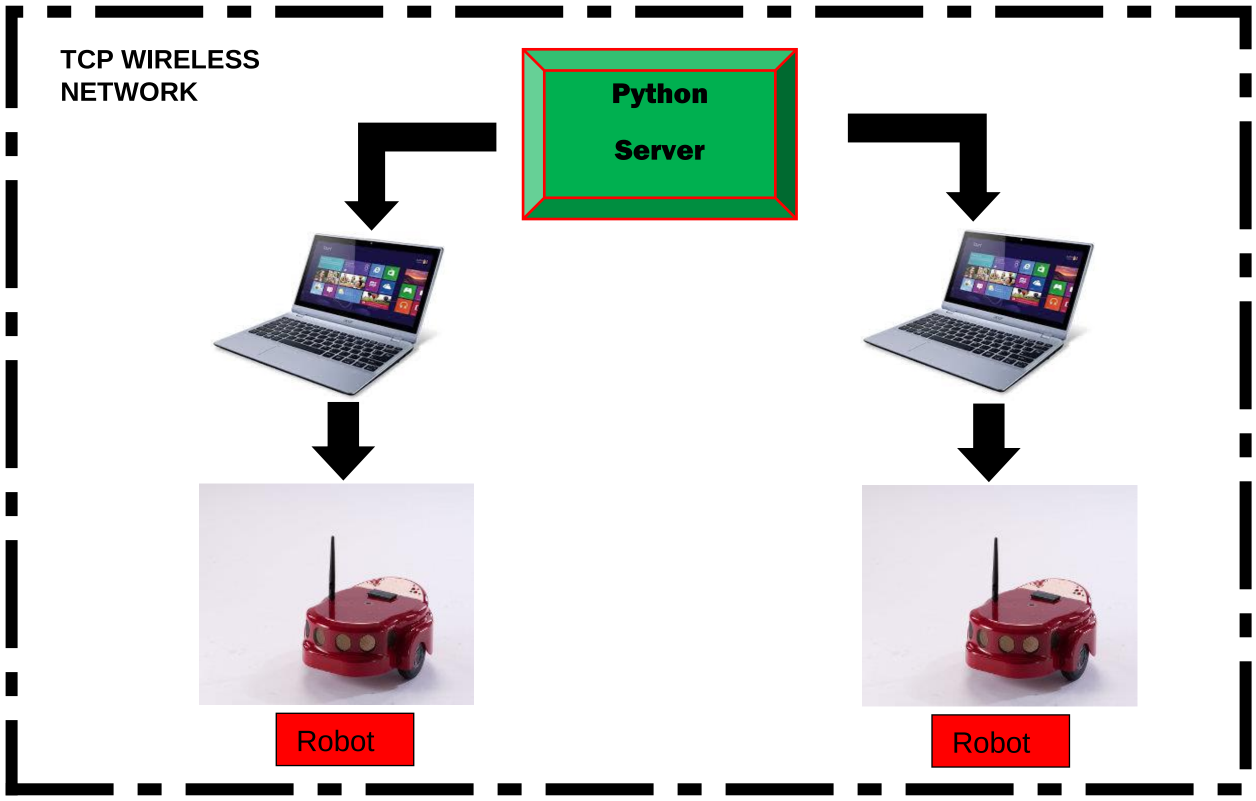


Figure 8: Information and Control Diagram

As shown above on Figure 8, the connection between Python Server and the robots. Information and programming control will be uploaded to server and then will be passed down to robots to do the movement and formation.

To achieve the theory of cooperation between the two robots, an aluminium strip has been chosen as a load to be carry by the robots while doing the movement and formation that has been programmed. This aluminium strip been chose because it is lightweight, thus a lightweight load may not harm or damage the robots.

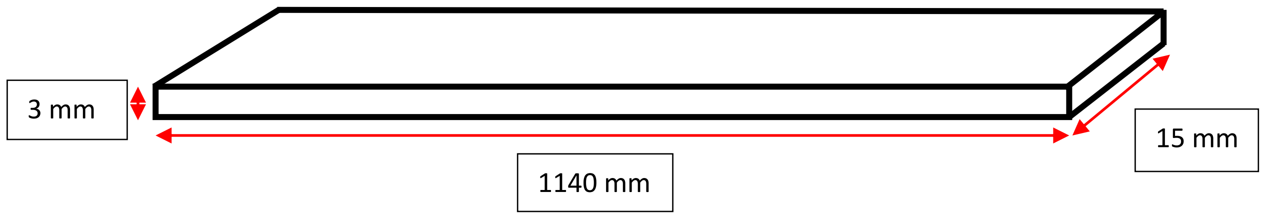


Figure 9: Aluminium Strip dimension

Figure 10: Aluminium Strip

4.3 Experiment Result and Discussion

The AmigoBot mobile robot are built on research purpose only, thus it is not suitable for industrial or actual field work application. Furthermore the result may different from other mobile robot.

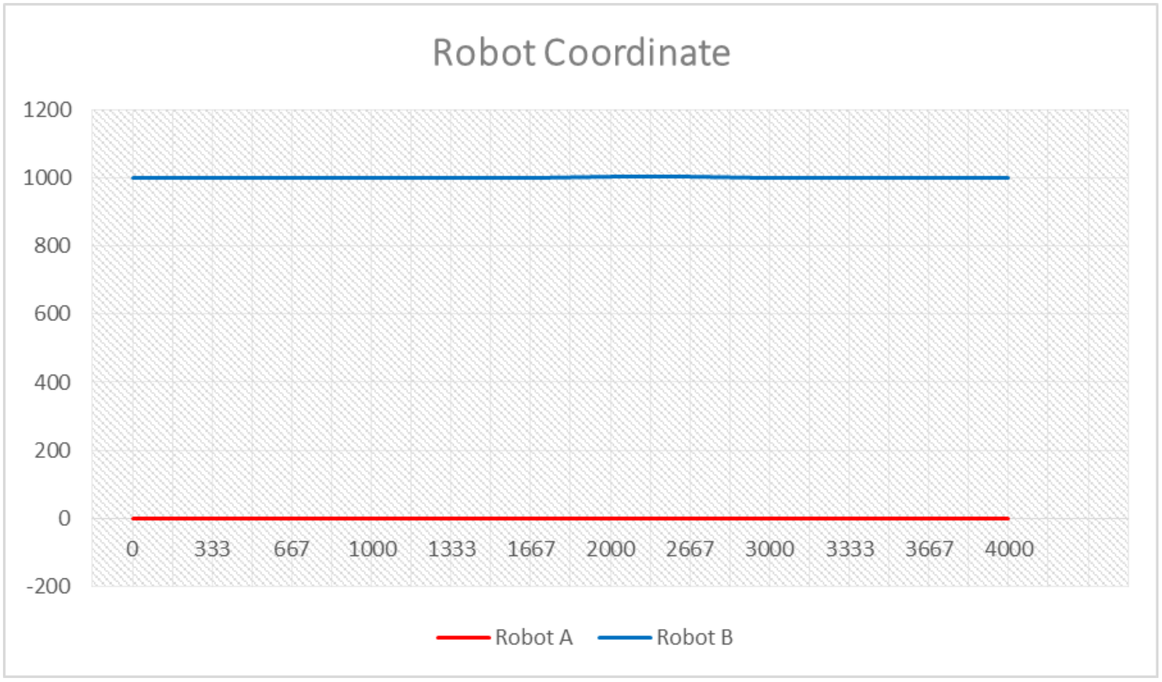
The programming control structures are also based on simple model structures, by a lot of variable that response to this experimental study, the action result also may not predicted. Variable such as internet connection to Python Server, execution of C++ programming and connections to robots are the most vital issues to be run properly without any problems. This AmigoBot mobile robot also equip with sonar sensor, which is high end for research application, but could also become one of the issues why the mobile robot does not performing formation because it detect a structures or obstacles and try to avoid it. Thus, the experiment area must be wide and clear without any obstacles.

The robots also could not detect any terrain defect, so it will not take any terrain defect as a possible causal of trajectory course defect. Moving either too fast or too slow tends to exacerbate the absolute position errors. The robots dead reckoning capability is a means of tying together sensor readings taken over a short period of time.

The graph plot represents the path taken by the robots as seen from top view. As the robots move to reach specified goal, real time coordinate for all the robots is captured using Python server. Coordinate information is used to plot the graph. Error calculation will be done based on the coordinate data instead of measuring it in actual environment.

4.3.1 Experiment 1

For the first experiment, two robots will carrying a load together and moving in a straight line for 4000 millimeters or 4 meters in a straight line at almost simultaneously. The coordinate for the first experiment is 10000 for the x-axis and 0 for y-axis.



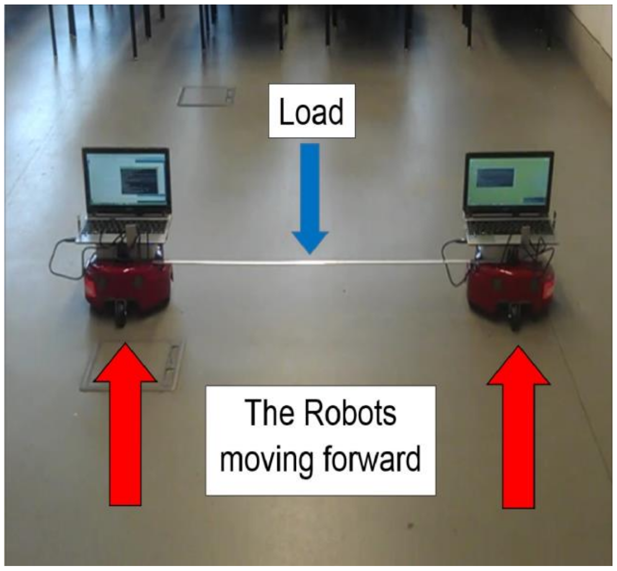


Figure 11: Experiment 1 result

As shown above Figure 11, a graph that show the movement of robots. Based on the graph, it shows no problem of robot movement carrying the aluminium strip as a load in 4 meter distance without having the load fell on the floor. This prove that the uniform velocity between robots to certain coordinate or destination is important.

As the experiment done, the aluminium strip still intact to robot without fell down to the floor. This result prove that the cooperation between the robots could carry the load from one point to another is successful.

	Robot A		Robot B	
	x (mm)	y (mm)	x (mm)	y (mm)
1	0	0	0	1000
2	333	0	333	1000
3	667	0	667	999
4	1000	-1	1000	999
5	1333	-1	1333	1000
6	1667	-1	1667	1001
7	2000	-2	2000	1003
8	2667	-1	2667	1002
9	3000	-1	3000	1000
10	3333	0	3333	999
11	3667	1	3667	999
12	4000	0	4000	1000

Table 2: Experiment 1 Data Coordination

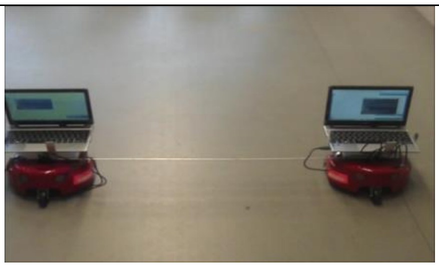
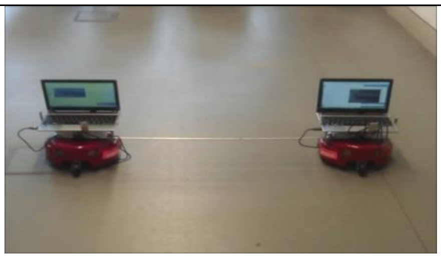
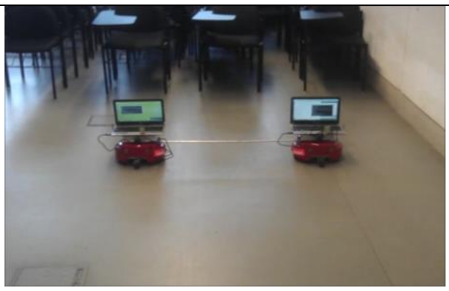
Number	Duration	Snapshot	Description
1	00:01		The beginning of experiment, everything in place and run the program.
2	00:08		Both robots start to move forward.
3	00:15		Continue to move forward with load still in place.
4	00:27		Reach 4 meter and the load still between robots.
5	00:30		Press 'Esc' button to close the program.

Table 3: Snapshot of Experiment 1 video.

4.3.2 Experiment 2

Experiment 2 are programmed to do more complex trajectory which is to do zig-zag movement while carry the load. Unfortunately the robots did not do the trajectory as programmed, and the Experiment 2 is consider as a fail. This failure caused either by the failure of programming or the incorrect connection upon the robots.

Investigation upon the program show that the list of command programming in C++ using the ARIA library is correct and valid, but still the robots fail to execute the zig-zag movement. This problem has many variable of causal, thus Experiment 2 did not run as planned. Some factors such as time and experience handling robot are the major constraint causing the experiment not happen.

4.4 Error Calculation

Error calculation is done to calculate either the robots movement are below targeted 10% fault tolerance. Fault tolerance tend start to happen when the targeted trajectory are too long or too far and causing the load that been carry fell on the floor.

Experiment	Mobile Robot	Target Coordinate (mm)	Maximum stray line (mm)	Fault Tolerance (%)

1	Robot A	x-axis = 4000
		y-axis = 0
			x-axis = 0
			y-axis = 2
				2.4

	Robot B	x-axis = 1000
		y-axis = 4000
			x-axis = 0
			y-axis = 3
				3.1

2	Robot A	N/A	N/A	N/A
	Robot B	N/A	N/A	N/A
			Average	2.75

Table 4: Fault Tolerance result

From table shown above, the fault tolerance that has been calculated from Robot A is 2.4%, slightly lower than Robot B in the Experiment 1. This may cause by the lagged command control from server to Robot B because Robot B is actually does not have any program on its own than Robot A. As for Experiment 2, no data could be recorded thus the average fault for mobile robot is 2.75%. It is considerably acceptable as this study targeted below 10% fault tolerance.

Chapter 5

Conclusion and Recommendation

Based on the experiment result, the targeted trajectory is a successful for Experiment 1 where the robots move in a straight line simultaneously and most importantly the load did not fell off while the robot move to targeted coordinate. But unfortunately the Experiment 2 where the robots where supposed to carry the load while moving in zig-zag movement did not happen. This failure may cause by faulty of programming code or connections between robots.

As for the error calculations or the fault tolerance, the robot doing great at Experiment 1 for achieving below 10% fault tolerance, which a good trajectory coordinate done even for a research purpose mobile robot. This prove that the robots will actually can do better if the programming control code are more specific and accurate.

Furthermore, for the betterment of future study, it is recommended to use far more advanced control programming method with more understanding and knowledge towards programming. It is also preferred to add more targeted coordinate by dotting every coordinate in line for more accurate trajectory. Other than that, an advance study towards the possibility of mobile robots and cooperative manipulators also need to be done to gain more knowledge and experience that is good to practice especially in the industrial field.

REFERENCES

1. Bahceci, E., Soysal, O., and Sahin, E., October 2003, *A Review: Pattern Formation and Adaptation in Multi-Robot Systems*. CMU-RI-TR-03-43, Robotics Institute Carnegie Mellon University Pittsburgh, Pennsylvania 15213. < http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1013622>

2. Hu H. et al., 1998 "Coordination of Multiple Mobile Robots via Communication," *In Proceedings of SPIE '98 Mobile Robots XIII Conference*, Boston, pp 94-103.

3. Kuppan Chetty, RM., Singaperumal, M. & Nagarajan, T., 2011, "Planning and Control Formation in Multiple Mobile Robots," *International Journal of Advanced Robotics System*. ISSN 1729-8806.

4. Kuppan Chetty, RM., Singaperumal, M. Karsiti N.B and Nagarajan, T., n.d., " State Based Modelling and Control of a Multi Robot Systems using SIMULINK

5. T. D. Barfoot and C. M. Clark,2004. *Motion Planning for Formations of Mobile Robots*, Robot. Auton. Syst., vol. 46, pp. 65-78.

6. Wu J., and Jiang Z., 2009, "Formation Control of Multiple Mobile Robots Via Switching Strategy," *In International Journal of Information and System Sciences Volume 5, number 2,* pg 210-218

7. Wang, P.K.C.,1992 "Navigation strategies for multiple autonomous robots moving in formation," *Journal of Robotic Systems 8 (2),* pp. 177-195. < http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=637948>

8. T. Balch and R.C. Arkin, 1998, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics and Automation*, pp. 926-939. < ieeexplore.ieee.org/xpl/freeabs_all.jsp?&arnumber=736776>

9. M. Breivik, T. Fossen, 2006, "Guided formation control for wheeled mobile robots," In9th IEEE Conference on Control, Automation, Robotics and Vision, Singapore, pp. 1_7.

10. Ghommam, J., Saad, M., and Mnif, F. 2008, "Formation path following control of unicycle-type mobile robots*",* *IEEE International Conference on Robotics and Automation Pasadena*, CA, USA. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4543495>

11. Balch T. and Hybinette M., 2000, "Behavior-based coordination of large-scale robot formations,". *Proceedings. Fourth International Conference on MultiAgent Systems*. < http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=858476

12. R. Beard, J. Lawton, and Hadaegh, 1999, "A coordination architecture for spacecraft formation control", *IEEE Transactions on Control Systems Technology* , pp. 777-790.

APPENDIX

```
This program has four modes to test the operation of the robot.

   1. Teleop mode:
      In this mode the robot can be controlled using the arrow
      keys on the connected computer. This is mostly for testing
      and convenience.
      It means that we don't have to pick up the robot and carry
      it back to a location if we want to rerun the test. We can
      just drive the robot to the location we want. The robot is
      quite heavy and not easy to carry.


   (NOTE: press the l key once the program starts up to activate

leader/follower)



   2. Leader mode:
      This mode is activated by the command line argument "-mode
      L". When the robot is in this mode it directly drives to
      the coordinates (3000, 3000) while transmitting its
      position to the Python server. If you want the robot to
      drive to a different location you can specify theX and Y
      coordinates of this new location using the -X and -Y
      command line arguments. For instance, if you want the robot
      to go to the point (5000, 5000) in leader mode you would
      run the program like so:

      actionGroupExample.DebugVC10 -mode L -X 5000 -Y 5000

      If you want to see this work open up MobileSim and observe
      the motion   of the robot.

   3. Follower mode:
      This mode is activated by the command line argument "-mode
      -F" When the robot is in follower mode, it will connect to
      the Python server to receive continuous updates on the
      position of the leader robot. It will then continuously
      change the goal of its ArActionGoto object to match the
      path of the follower. The follower mode cannot be run
      without the server and a leader running at the same time.
```

The leader mode can however, be run independently.

4. DrawSquare mode:
   This mode is activated by pressing the s key after the program starts running. Once activated the robot will move in a square of side 10m.

Details on setting up and running the Python server are in the README file.

--SETTING UP THE NETWORK--

For the robots to be able to exchange information over the network, all of them have to be connected to the same wireless network.
Once you are sure that the laptops on the robots are connected to your wireless network you need to find the IP addresses of all the laptops.
To find the IP of a computer open up the command prompt and type in ipconfig and press enter.
Your computer's IP address will be listed under the details of the wireless adapter your laptop uses to connect to the network.
On most modems it's something like "192.168.1.4". Once you have all the IP addresses, write them down. This might be useful later.

--SETTING UP THE SERVER--

There are two ways to set up a server:

1. Use a separate laptop as a server.
Install Python 2.7 in your laptop. (NOTE: THE PROGRAM DOES NOT WORK WITH PYTHON 3.3, THE LATEST VERSION. IT ONLY WORKS WITH PYTHON 2.7. MAKE SURE YOU HAVE 2.7).
Once you have it installed, open up the Python interpreter (called IDLE).
Click File and open the Server.py script which is inside the ZIP file.
Run the program.
As soon as the program runs, it will ask you which port to use. Enter 27015.
Sometimes when you run the program a second time it will show an error saying that the port is already in use.
If this is the case, stop running the program, start the program again and use a different port (like 27016).
MAKE SURE THAT THIS PORT NUMBER IS THE SAME AS THE PORT NUMBER YOU GIVE THE ROBOT PROGRAMS RUNNING ON THE TABLETS.

Once you enter the port number, the program will start listening for connections and act as a server.
It is now ready accept incoming connections from a laptop attached to the robot.
When the program first connects it will Display the message "Red leader Checking in".
This will let you know that the robot is connected and ready to go into leader/follower mode.

Once you run the programs on the robot, the python server will start displaying a stream of data that it received from the program.
The program will also start displaying a stream of data it received form the Python server.

--INSTRUCTIONS ON RUNNING THE PROGRAM—

1. Compile the program in Visual C++: Compiling is a bit tricky. Setting up the Aria libraries for compilation on a new project is quite tricky.
So instead open up one of the existing projects in the C:\Program Files\MobileRobots\Aria\examples folder.

Opened up the actionGroupExample project. These projects already have the settings modified.
So all you need to do is click compile. Replace the existing code with the code in the robot_code.cpp file and compile.
The executable compiled program will be created in the C:\Program Files\MobileRobots\Aria\binfolder.
If you chose the actionGroupExample project, the executable will be named actionGroupExample.
DebugVC10. This is the executable we have to run.

2. Running the Program: To test the program with the simulator or a robot, you have to run the program from the command line.
To run the program from the command line, open up the command line in Windows. Then navigate to the bin folder in the command line using cd.
Type in "cd C:\Program Files\MobileRobots\Aria\bin" to navigate to the correct folder.
To run the program type in "actionGroupExample.DebugVC10" into the command line.
To run the program in leader mode and make it go to a coordinate point type:

```
      "actionGroupExample.DebugVC10 -mode L -X 4000 -Y 4000"
```

The 4000 is the coordinate point in millimeters.
The program connects to the simulator and starts up in Teleop mode.
Press 'l' on the keyboard to activate the leader/follower mode. The bot should drive in a straight line towards the specified point. If you want to connect to a Python server at a specific IP address (say 192.168.1.8) run:

```
"actionGroupExample.DebugVC10 -rs 192.168.1.8 -mode L -X 4000 -Y 4000"
```

To run the robot in follower mode type:

```
"actionGroupExample.DebugVC10 -rs 192.168.1.8 -mode F"
```

You can also specify the port to be used for communication. MAKE SURE THAT THIS PORT IS THE SAME AS THE ONE YOU GIVE THE PYTHON SERVER! IF NOT THE PROGRAM WILL NOT BE ABLE TO CONNECT. If you want to specify a port type:

```
"actionGroupExample.DebugVC10 -rs 192.168.1.3 -mode L -X 3000 -Y 3000
-port 27015"
```

```
#include "Aria.h"
#include <stdio.h>
#include <iostream>
#define SERVER_IP "127.0.0.1"
#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLEN 22
#include <string.h>
#include <sstream>

using namespace std;
//Variable that determines if the robot is in leader mode or not
char *L = "L";
char *F = "F";
char *leader_flag = L;
int modeFlag = 1;
int switchFlag = 0;

//Target coordinates when the robot is in follower mode. This keeps changing
double X_goal = 4000;
double Y_goal = 4000;
char * pch; char *end;
char *s; char *s2;
ArActionGoto GoTo("drive to target", ArPose(X_goal, Y_goal), 100, 300, 150, 7);

//Target coordinates for when the robot is leading. This does not change.
double leader_goalX = 3000;
double leader_goalY = 3000;

//Side length of the square for the drawSquare action group
double side_length = 10000;

//Network data
char *sendbuf = "Bot1 ACK";
char recvbuf[DEFAULT_BUFLEN];
int iResult;
int recvbuflen = DEFAULT_BUFLEN;
///Strings that will contain the conversions
string xcord; string ycord;

//Function to convert from number to string
template <typename T>
  string NumberToString ( T Number1, T Number2, T Number3 )
  {
     ostringstream ss;
     if(strcmp(leader_flag, F)==0) ss <<
Number1<<","<<Number2<<","<<Number3<<".";
        if(strcmp(leader_flag, L)==0) ss
<<"L"<<Number1<<","<<Number2<<","<<Number3<<".";
     return ss.str();
  }


void updateGoal(string str)
{
    size_t found1 = str.find(",");
    X_goal = (double)atoi((str.substr(0, found1)).c_str());
    size_t found2 = str.find(",", found1+1);
    Y_goal = (double)atoi((str.substr(found1+1,found2-found1)).c_str());
       if(strcmp(leader_flag, F)==0)
       {
              GoTo.setGoal(ArPose(X_goal,Y_goal));
              Goto.setSpeed(400);
       }
       if(strcmp(leader_flag, L)==0)
       {
              GoTo.setGoal(ArPose(leader_goalX, leader_goalY));
              GoTo.setSpeed(300);
       }
/*     if((strcmp(leader_flag, F))==0)
       {
              GoTo.setGoal(ArPose(X_goal+500,Y_goal+500));
       }
       if(strcmp(leader_flag, L)==0)
       {
              GoTo.setGoal(ArPose(leader_goalX, leader_goalY));
       }
    //printf("\nX: %d\n", X_goal);
*/    //printf("Y: %d\n", Y_goal);
}

//ArAction classes
class ActionCom : public ArAction

{
public:
       //Constructor
       ActionCom(ArTcpConnection *data_link, ArRobot *robot);
       //Destructor. Does not need to do anything
       virtual ~ActionCom(void) {};
       //Called by the action resolver to obtain the action's requested behaviour
       virtual ArActionDesired *fire(ArActionDesired currentDesired);
       //Store the robot pointer and it's ArSonarDevice object, or deactivate
this action if there is no sonar
       virtual void setRobot(ArRobot *robot);
       int x_goal, y_goal;
protected:
       //the sonar device object is obtained from the robot by setRobot()

       ArRangeDevice *mySonar;
       ArTcpConnection *tcpptr;
       ArRobot *robotptr;
       int x_pos; int y_pos; int heading;
       double obsDist, angle;
       /*  Our current desired action: fire() modifies this object and returns to
the
            action resolver a pointer this object. This object s kept as a class
member
               so that it persists after fire() returns (otherwise fire() would
have to
               create a new object each invocation but would never be able to
delete that
               object).
```

```
	*/
	ArActionDesired myDesired;
};

ActionCom::ActionCom(ArTcpConnection *data_link, ArRobot *robot) : ArAction("Go")
{
	//This is the constructor. Some data setup is needed
	//I have waited so long to do this.
	data_link->write("Red Leader Checking in!", DEFAULT_BUFLEN);
	//Set an internal pointer to point to the TCP connection
	tcpptr = data_link;
	//Set an internal pointer to the robot
	robotptr = robot;
	//Get the curent pose of the robot.
	x_pos = (robotptr->getX());
	y_pos = (robotptr->getY());
	heading = robotptr->getCompass();
	printf("Sending data to server.");
}

void ActionCom::setRobot(ArRobot *robot)
{
	ArAction::setRobot(robot);
	mySonar = robot->findRangeDevice("sonar");
	if(robot==NULL)
	{
		ArLog::log(ArLog::Terse, "ActionCom: Warning! I found no sonar.
Deactivating");
		deactivate();
	}
}

/*
	This fire is the whole point of the action.
	currentDesired is the combined desired action from other actions
	previously processed by the action resolver. In this case, we're
	not interested in that, we will set our desired forward velocity
	in the myDesired member and return it.

	Note that myDesired must be a class member since this method will
	return a pointer to myDesired to the caller. If we had declared

	the desired action as a local variable in this method the pointer we
	returned would be invalid after this method returned.

*/

ArActionDesired *ActionCom::fire(ArActionDesired currentDesired)
{
	//Get the current pose of the robot
	x_pos = robotptr->getX();
	y_pos = robotptr->getY();
	heading = robotptr->getCompass();
	obsDist = (robotptr->checkRangeDevicesCurrentPolar(-70, 70, &angle) -
robotptr->getRobotRadius());
	//I'm not resetting the action because it does not need to be done
```

```
        //Figure out if this is ok during testing.
        //myDesired.reset();
        //myDesired.setVel(0);
        if(obsDist < 450) switchFlag = 1;
        //Send the pose as a string over the TCP connection
        tcpptr->write(NumberToString(x_pos, y_pos, heading).c_str(),
DEFAULT_BUFLEN);
        //Get the position of the leader
        tcpptr->read(recvbuf, DEFAULT_BUFLEN, 5);
        printf("Received: %s\n", recvbuf);
        //Convert the message to a string
        string str = (string)recvbuf;
        //cout<<"\nString: "<<str<<"\n";
        //Update the goal of the robot
        //if(recvbuf[0] == 'S') modeFlag ^= 1;
        updateGoal(str);
        //cout<<"\nX: "<<X_goal<<" Y: "<<Y_goal<<"\n";
        return &myDesired;
}

//Main function
int main(int argc, char** argv)
{
        //Initialize the ARIA libraries
    Aria::init();
        //The ArArgumentParser object processes the command line arguments given
to the program.
        //It does most of the processing for you and allows us to easily extract
the command line
        //options for use inside the program.
    ArArgumentParser argParser(&argc, argv);
        //The variable the tells the program which port to use for the TCP
connection
        int conPort = 27015;
        //This command line argument tells the program the IP address of the
server
        char* server = argParser.checkParameterArgument("-rs");
        //This command line argument tells the program which mode to operate in.
Leader or follower.
        char* mode_flag = argParser.checkParameterArgument("-mode");
        char* port = argParser.checkParameterArgument("-port");
        if(!port) conPort = 27015;
        if(port) conPort =(int) atoi(port);
        if(!mode_flag) leader_flag = L;
        //else if(strcmp(mode_flag, "F")) leader_flag = F;
        //else if(strcmp(mode_flag, "L")) leader_flag = L;
        else leader_flag = mode_flag;
        //Sets the IP address to localhost if no IP is specified
        if(!server) server = "localhost";
        //The global X and Y coordinates to which the robot must go to in Leader
mode.
        char* x_togo = (argParser.checkParameterArgument("-X"));
        char* y_togo = (argParser.checkParameterArgument("-Y"));
        //if(x_togo) X_goal = (double)(atoi(x_togo));
        //if(y_togo) Y_goal = (double)(atoi(y_togo));
        if(x_togo) leader_goalX = (double)(atoi(x_togo));
```

```
       if(y_togo) leader_goalY = (double)(atoi(y_togo));
       cout<<"\nX: "<<leader_goalX<<" Y: "<<leader_goalY<<"\n";
       //Tells the robot to go to the leader goal by changing the destination of
the
       //ArActionGoto object to the coordinates given in the command line.
       //If no command line arguments were given the robot will just go to 5000,
5000
       //since these are the default values of leader_goalX and leader_goalY
       GoTo.setGoal(ArPose(3000, 3000));
       //This is an ArSimpleConnector object. This object takes care of
connecting to the robot
       //using the serial port and telling the robot what to do. The high level
programming takes care
       //of the rest.
    ArSimpleConnector con(&argParser);
       //This is the main and most important object in the entire program. The
ArRobot object represents the
       //robot in the program. Any changes made to this robot object will be
reflected in the actual robot.
       //Modifying the robot object's linear velocity, rotational velocity or
heading will change the actual
       //robot's linear velocity, rotational velocity or heading.
    ArRobot robot;
       //This object represents the sonar object on the robot in the program. Any
changes made to this object
       //in the program will be reflected on the actual sonar.
    ArSonarDevice sonar;
       //This is a declaration for a new ArTCPConnection object. This object
handles all the difficult handshaking
       //and signalling necessary to communicate with another computer over a
TCP/IP connection. This object is
       //responsible for communicating with the Python server and sending it
information about the robot's current
       //position and also receiving the leader's position.
       ArTcpConnection data_link;
       //The open() funtion opens a connection to the python server. The "server"
variable contains the IP address
       //of the server. If you are testing the program on a simulator the server
is just "localhost" or "127.0.0.1"
       //which is the loopback address. If the program is running on a wireless
network created by a router, then
       //the server is thre local IP address of the computer which is running the
server. The variable "server" is
       //specified by the -rs command line argument. Make sure that all firewalls
are turned off to ensure that
       //TCP connections are not refused!
       //The number 27016 is the port which the TCP connection uses. This can be
any number between 1 and 65536 but
       //ports with low numbers are ususally reserved for other important net
services. So it is best to use a nice
       //high number like 27015.
       //Replace 127.0.0.1 with the IP address of the server.
       data_link.open(server, conPort);

    argParser.loadDefaultArguments();
    if(!Aria::parseArgs() || !argParser.checkHelpAndWarnUnparsed())
    {
        Aria::logOptions();
        return 1;
    }

       //This is the action group for the teleoperation mode of the robot
       //Notice how the actions are listed in order of priority.

    /* - the action group for teleoperation actions: */
    teleop = new ArActionGroup(&robot);
    // don't hit any tables (if the robot has IR table sensors)
    teleop->addAction(new ArActionLimiterTableSensor, 100);
    // limiter for close obstacles
    teleop->addAction(new ArActionLimiterForwards("speed limiter near",
                                      300, 600, 250), 95);
    // limiter for far away obstacles
    teleop->addAction(new ArActionLimiterForwards("speed limiter far",
                                      300, 1100, 400), 90);
    // limiter so we don't bump things backwards
    teleop->addAction(new ArActionLimiterBackwards, 85);
    // the joydrive action (drive from joystick)
    ArActionJoydrive joydriveAct("joydrive", 400, 15);
    teleop->addAction(&joydriveAct, 50);
    // the keydrive action (drive from keyboard)
    teleop->addAction(new ArActionKeydrive, 45);


       //THIS ACTION GROUP IS NOT USED IN THE PROGRAM
    /* - the action group for follow actions: */
    follow = new ArActionGroup(&robot);

       //Transmit location and receive goal
       follow->addAction(new ActionCom(&data_link, &robot), 100);
    // if we're stalled we want to back up and recover
    follow->addAction(new ArActionStallRecover, 99);
    // react to bumpers
    follow->addAction(new ArActionBumpers, 75);
    // turn to avoid things closer to us
    follow->addAction(new ArActionAvoidFront("Avoid Front Near", 225, 0), 50);
    // turn avoid things further away
    follow->addAction(new ArActionAvoidFront, 45);
    // keep moving
    follow->addAction(&GoTo, 43);


       //This is the most important action group in the program. This makes the
robot
       //move in leader mode or follower mode depending on the command line
arguments
       //NOTE: THIS IS THE actiongroup that is used for ALL THE THINGS. THE
FOLLOW
       //ACTIONGROUP IS NOT INVOLVED AT ALL.
       //The action group for the leader mode operation
       leader = new ArActionGroup(&robot);

       //Highest priority for recovering froma stall.
```

```
        leader->addAction(new ArActionStallRecover, 100);
        //Next highest priiority for responding to bumpers.
        leader->addAction(new ArActionBumpers, 75);
        //Avoiding obstacles detected by the sonar that are close by.
        leader->addAction(new ArActionAvoidFront("Avoid Front Near", 225, 0), 50);
        //Avoiding obstacles that are far away.
        leader->addAction(new ArActionAvoidFront, 45);
        //Going to the goal.
        leader->addAction(&GoTo, 43);
        //Connection test
        leader->addAction(new ActionCom(&data_link, &robot),44);


        //This is the action group that makes the robot move in a square.
        drawSquare = new ArActionGroup(&robot);

        //Highest priority is given to recovering from stalls.
        drawSquare->addAction(new ArActionStallRecover, 100);
        //Next highest priiority for responding to bumpers.
        drawSquare->addAction(new ArActionBumpers, 75);
        //Avoiding obstacles detected by the sonar that are close by.
        drawSquare->addAction(new ArActionAvoidFront("Avoid Front Near", 225, 0),
50);
        //Avoiding obstacles that are far away.
        drawSquare->addAction(new ArActionAvoidFront, 45);
        //Sides of square
        //Traversing each side of the square is given with decreasing priority,
        drawSquare->addAction(new ArActionGoto("Side 1", ArPose(side_length, 0,
90), 100, 400, 150, 7), 44);
        drawSquare->addAction(new ArActionGoto("Side 2", ArPose(side_length,
side_length, 180), 100, 400, 150, 7), 43);
        drawSquare->addAction(new ArActionGoto("Side 3", ArPose(0, side_length,
270), 100, 400, 150, 7), 42);
        drawSquare->addAction(new ArActionGoto("Side 4", ArPose(100, 100, 270),
100, 400, 150, 7), 41);



    /* - use key commands to switch modes, and use keyboard
     *   and joystick as inputs for teleoperation actions. */

    // create key handler if Aria does not already have one
       //The keyHandler keeps checking for keyboard input and detects it
       //so that the program can change modes.
    ArKeyHandler *keyHandler = Aria::getKeyHandler();
    if (keyHandler == NULL)
    {
        keyHandler = new ArKeyHandler;
        Aria::setKeyHandler(keyHandler);
       robot.attachKeyHandler(keyHandler);
    }

    // set the callbacks
    ArGlobalFunctor teleopCB(&teleopMode);
    ArGlobalFunctor followCB(&followMode);
       ArGlobalFunctor leaderCB(&leaderMode);
       ArGlobalFunctor squareCB(&drawSquareMode);

       //This is not used
    keyHandler->addKeyHandler('F', &followCB);
    keyHandler->addKeyHandler('f', &followCB);

       //If t or T is pressed the robot enters teleop mode. You can control the
robot
       //using the arrow keys on the computer in this mode.
    keyHandler->addKeyHandler('t', &teleopCB);
    keyHandler->addKeyHandler('T', &teleopCB);

       //Pressing l or L puts the robot in leader/follower mode. This makes the
robot
       //behave as a leader or a follower depending on the command line
arguments. This
       //where the magic happens.
       keyHandler->addKeyHandler('l', &leaderCB);
       keyHandler->addKeyHandler('L', &leaderCB);

       //Pressing s or S puts the robot in drawSquare mode. This will make the
robot move in a square
       //of side length 10 m
       keyHandler->addKeyHandler('s', &squareCB);
       keyHandler->addKeyHandler('S', &squareCB);

    // if we don't have a joystick, let 'em know
    if (!joydriveAct.joystickInited())
        printf("Note: Do not have a joystick, only the arrow keys on the keyboard
will work.\n");

    // set the joystick so it won't do anything if the button isn't pressed
    joydriveAct.setStopIfNoButtonPressed(false);

       //Setting up the robot as the client

    /* - connect to the robot, then enter teleoperation mode.  */
       //This line of code connects the sonar device to the robot object.
       //In the real world this makes sure that the robot is getting readings
from
       //the sonar. This is necessary for obstacle detection to work.
    robot.addRangeDevice(&sonar);

       //Error handling
    if(!con.connectRobot(&robot))
    {
        ArLog::log(ArLog::Terse, "actionGroupExample: Could not connect to the
robot.");
       Aria::exit(1);
    }

       //Enable the motors of the robot so that the robot can move.
    robot.enableMotors();
       //As soon as the program starts up, it enters teleop mode. For
convenience.
    teleopMode();
```

```
        //Run the processing cycle of the robot.
    robot.run(true);

    Aria::exit(0);
        data_link.close();
}
```