5-1-2009

Implementation of Robot Arm Networks and Experimental Analysis of Consensus-Based Collective Motion

Daniel Scott Stuart
Utah State University

IMPLEMENTATION OF ROBOT ARM NETWORKS AND EXPERIMENTAL ANALYSIS OF CONSENSUS-BASED COLLECTIVE MOTION

by

Daniel Scott Stuart

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Wei Ren Major Professor	Dr. YangQuan Chen Committee Member
Dr. Brandon Eames Committee Member	Dr. Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2009

Abstract

Implementation of Robot Arm Networks and Experimental Analysis of Consensus-Based Collective Motion

by

Daniel Scott Stuart, Master of Science

Utah State University, 2009

Major Professor: Dr. Wei Ren
Department: Electrical and Computer Engineering

Within the field of multi-robot control, there is a large focus in research involving consensus. In this thesis two parts will be studied. The first development of this thesis is a consensus-based robot arm platform. To implement, two robotic arms are developed and studied. The most effective robot arm is then utilized to create a robot arm network testbed. Consensus is used to coordinate several robot arms and decentralize system computation. The research explores a platform to facilitate consensus on a group of robotic arms.

The second development is in Cartesian coordinate collective motion. This collective motion control combines consensus through coupling of Cartesian coordinates. The controller is presented with simulation and experimental validation. Integration of both parts of the thesis is then discussed in application. An example is provided to demonstrate usefulness. In conclusion, this thesis provides more control to a system of ground robots using collective motion and consensus-based robot arms.

(89 pages)

To my parents....

Acknowledgments

I would like to acknowledge and thank my major professor, Dr. Wei Ren, for his unwavering support, patience, and guidance. I also would like to acknowledge the other members of my committee, Dr. YangQuan Chen and Dr. Brandon Eames. I have learned a great deal in skill, education, and life, both in and outside the classroom, from these members and it is greatly appreciated. I would like to acknowledge the members of both the CSOIS and COVEN labs from whom I have learned work ethic and enthusiasm for research. It has been a pleasure to work with so many interesting and wise people who have the same level of love for robots as I do. Finally, I absolutely want to acknowledge my parents who have stood by my side and supported me through anything and everything.

Daniel Scott Stuart

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the field of robotics, the area of autonomous multi-agent robots has become one of increasing interest and exploration in more recent years. While single mobile robots have been used with huge success in solving societal problems, they fight the issue of having need for huge computational power, expensive construction, and time. In solving any problem, if a single robot is used and that single robot breaks down or fails for any reason, the problem is unsolved, and a great deal of cost must be spent to get the robot back up. In that frame of mind set, where one robot is good at solving problems, a group of robots can be even better. In a group of robots, a single failure of anyone robot, will not mean that the problem solving has to necessarily stop. Many smaller and inexpensive robots can be used to replace one large and expensive robot. If one ground robot carrying a large tank of water is used to fight a fire, it must not go down, and it must spend a great deal of time navigating around all parts of the fire to put it out. Whereas a group of small robots with tanks can distribute and cooperate their effort on the fire until all parts are out. If one or two robots happen to fail during the process, the group simply must reorganize and redistribute efforts. In either case, the problem can be solved and system robustness remains higher. An additional benefit of a large group of robots in a system instead of one is communication. In a group of robots, the spread of communication data can be sent long distances through multi-path hops along robots within the group. This sharing of data not only preserves robustness of the communication but also allows for long distance communication through agents working as relays, a task that would not be possible with one robot and would require expensive communication hardware.

Early attempts at group robotic control focused on a centralized hub that would dictate

instructions to each individual agent. These schemes were divided into leader-follower architectures [1, 2] as well as virtual architectures [3]. However, the use of a centralized control negates many of the benefits that can be gained in computation as well as system robustness by having multiple robots. In such a centralized system, each agent is merely a puppet, which means that the central hub handles all of the computational load and the whole system stops if the central hub is damaged or lost. To utilize the above benefits of a large group of robots, it makes more sense to decentralize the control, relying on each robot in the group to provide some computation power. This means a loss of one robot only causes a decrease in computational power and not system death. Such systems consist of behavior based control [4]. However, behavior networks suffer the inability to maintain strict formations or collective motions. One promising approach to decentralization while keeping stricter motion and formations is the use of consensus. Consensus is a concept originating in computer science in fault tolerance. In the presence of a fault, consensus allows a group of modules to determine via agreement the best course of action. In a multi-agent mobile robot system, this translates to an agreement without a centralized congregation. Consensus has been placed in tandem with earlier virtual architectures to create a consensus based multi-agent control theory [5–7].

1.2 Contributions

1.2.1 Single Robot Arm Development

In previous work, consensus based schemes have been applied to many forms of mobile agents such as ground and air robot vehicles. The use of consensus to create formations and converge on problem issues is powerful. Unmanned aerial vehicles (UAVs) can be used with cameras or chemical actuators to cover large areas in observance or active tasks. However, ground vehicles have limited use without the addition of some form of actuator other than a camera. The current multi-robot testbed used for research in our lab is based on the AmigoBot mobile robot platform [8]. The current platform allows for the testing of consensus based algorithms on position and velocity control of the ground robots. However,

they do not include any manipulators or actuators limiting research to ground trajectories and their ability to solve problems. In this thesis, two robot arms are designed to operate off of the current mobile ground platform. This objective will serve not only to provide each ground robot with more flexibility, but it will also serve as a foundation for the robot arm network testbed discussed within the next sub-section. Each robot arm is explored in its capability and a final robot arm design is chosen and pushed through to a complete conclusion. The final robot arm can be controlled through tele-operation from a static computer within the lab or it can be controlled from the on-board computer on each ground robot. The final robot arm has four degrees of freedom including rotational, shoulder, elbow, and wrist joints. The robot arm also has a gripper that it can use to manipulate objects within its environment. In this thesis, software functions were written to provide useful application in future research and for current research goals in robot manipulation. The joints of each arm can be individual controlled by both open-loop control or closed-loop proportional-integral-derivative (PID) control. The robot arm also has software written for implementing inverse kinematics, allowing for a three space position of the end effector to be given for the robot arm to meet. The robot arm software also includes written functions to provide individual angle feedback of each arm joint. Finally, this thesis provides software to allow for each robot arm to be driven as if it was a motor, thus allowing for many of the consensus algorithms which rely on single integrator kinematics to be used.

1.2.2 Robot Arm Network Testbed

In a system of mobile robots being controlled through consensus, the use of consensus to control not only position trajectory but also the states of on-board actuators is new. In this thesis, the robot arms previously described are combined into a testbed for researching consensus control algorithms in groups of static or mobile robot arm manipulators. The idea of control of multiple static and mobile manipulators is not new [9, 10]. However, using consensus as a structure to synchronize or drive mobile robot arms is new. The ability to synchronize robot arm manipulators within a network of mobile robots has many applications. Tasks such as filming a room from different perspectives in a similar manner

or jointly carrying an object across the room can be simply accomplished using consensus. Passing objects from one manipulator to another or fighting a fire jointly require that manipulators work in unison to be the most effective. While there are an endless amount of possibilities, a system of distributed mobile robotic arm manipulators must first be created. In this thesis, a platform test bed is created to facilitate the research into controlling a network of robot arms. To allow for consensus to work, each agent in the system must be able to be controlled using single integrator kinematics, and feedback of the agents state must be able to be shared with other agents in this system. To share information with other robot arms within the networks, the use of existing network protocols and functions is derived to allow each robot arm to set up a server in which it can share its individual joint angles with any requesting robot arm or robot on the system. The only requirement to retrieve data from a robot arm, is that both sides know the appropriate port number, protocol address, and protocol format.

The robot arm network must also be able to drive each robot arm joint with given control output from each consensus algorithm. In this case, the control signal is velocity which was implemented for each robot arm in previous parts of the thesis as described earlier. The robot arm network facilitates the use of a network by running its software off of a static host computer with a network connection or the on-board computers on the AmigoBot platform which have wireless network sharing capability. Within the research lab, the communication between any size group of robots within the local wireless network is always strong. However, much of the consensus research requires the implementation of a communication topology, where some agents may not be able to connect with other agents, or they may have limited connectivity with the system. To be able to test this realm of the research, the robot arm network software also includes facilities to prescribe a simulated network connection topology. In this way, the user of the software can implement how each arm can talk with or interact with the rest of the robots within the system. This allows for further possibilities in research of consensus algorithms. In this thesis, a general software testbed platform has been created and a few small demonstrations of its use are shown to

provide a conclusion of its future possibilities within the lab's research goals.

Finally, a robot network should also be able to include external references from outside sources in driving where the synchronization and consensus of the robot manipulator arms go. To allow for this, this thesis discusses software implementations that can work on a static computer or host computer external to the system, but connected to the network. In this software, the reference program can then provide another source of control reference that some or all of the robot arms can know in driving the state for the whole system.

1.2.3 Cartesian Coordinate Coupled Control

Current research theory within the lab surrounds around the idea of Cartesian coupling and collective motion [11]. Within the current consensus-based control theory, many of the control schemes have been validated to be true on physical discrete time-based mobile robots [12]. However, the validation of current theories of combining leaderless collective motion and Cartesian coupling have not been tested on physical discrete time mobile platforms. Synchronization and collective motion can be found all around nature. Large groups of fish or birds move as a group, but rely on their interdependent positions to move individually, causing a collective motion of the group. In general consensus, the synchronization of the agents is decoupled. That is, each coordinate state whether it be x,y,z,angle,etc., is individually consented together with those decoupled states of the other agents. In Cartesian coordinate coupled collective motion, the states of the trajectory of each robot (x,y,z) are coupled together in a consensus control through the use of a rotation matrix. The use of a rotation matrix in conjunction with general consensus algorithms creates the ability to control the motion and stability of a system through the control of the rotation angle of the rotation matrix. By varying that angle, the system of robots cyclically pursue each other while converging in a stable manner, diverging in an ultimately unstable manner, or orbiting in a marginally stable manner. This introduction of control allows for the system of robots to move in collective motions of either orbits or logarithmic spirals in and out. In this thesis, I implement several Cartesian coupling theories in the currently designed MobileRobots, Inc AmigoBot platforms [8].

The first controller implemented on the AmigoBot platform for experimental validation is consensus using Cartesian coupling through a single integrator kinematic system. The AmigoBot system is a single integrator dynamic system which means that it can be controlled by driving the vehicle's velocity. In this thesis, the continuous time controller [13] is studied first before the discrete time version is implemented on the actual ground robot platforms. This controller is implemented on Matlab using a strict single integrator kinematic system, and the results are then presented as a reference to compare with the results of the experimental validation.

The second controller implemented on the AmigoBot platform for experimental validation is consensus using cartesian coupling through a double integrator dynamic system. Many robot platforms are best driven and controlled using double integrator dynamics such as changing motor torques or the present acceleration of the vehicle. Controlling the double integrator dynamics of a system designed to run using double integrator dynamics, thus provides for better control. In this thesis, the continuous time controller [14] is studied first before the discrete time version is implemented on the actual ground robot platform. This controller is implemented on Matlab using a strict double integrator dynamic system, and the results are then presented as a reference to compare with the results of the experimental validation. In this case, the AmigoBot testbed is best controlled with velocities in a single integrator kinematic form. To best facilitate the testing and validation of double integrator dynamics, the system therefore must be modified to except double integrator control signals and then convert them to single integrator kinematics for actuation. This becomes acceptable since, the feedback to the double integrator control algorithms still utilizes the states that are needed for the control actuation.

The purpose of this thesis is to validate the simulation results and provide a structure for which present and future study in Cartesian coupled collective motion control can be implemented. Results are provided, for both controllers in simulation form and from control experimentation on the actual physical robot platform. The experimental results coincide properly with those of the simulation and provide validation that the controllers work

properly even in a discrete form, provided that the sampling times in the discrete case meet with stability as with any discrete controller algorithm.

1.2.4 Future Combination and Contribution

To provide some study on the usefulness of the research discussed in previous sections, the research is combined in a simulated example. One problem considered within group robotics is the issue of coverage or filling up an area with a group of robotics to completely focus on a problem. Whether the issue is a set of aerial vehicles patrolling an area or a set of ground vehicles searching a room, there needs to be an aspect of full coverage. Two similar problems are considered for this section. Both involve covering a large spatial area in a quasi-uniform manner. In stereo vision, an object cannot be filmed from only one perspective. If a room is to be mapped as to gather 3-dimensional data, the room must be covered from various offsets of a viewing angle. In the same manner, if a room is to be rid of some gas, fire, or chemical diffusion, it is better not to cover the same exact area each time as the robot arms sprays a room. As with any problem, multiple robots offer more efficient use of resources while maintaining robustness of solving the problem even through vehicle loss. The purpose therefore of this section is to present one option of robotic motion of both ground vehicles and robot arms that could be adapted to accomplish these goals. This section will utilize the concept of the robot arm network testbed and the ground vehicle collective motion study. Using a simulation of four ground robots with attached robot arms, a possible implementation of full coverage of a room through robot arm and ground vehicle motion will be shown.

1.3 Organization

The following aspects of the thesis are organized to develop the topics and research provided previously. In Chapter 2 the aspects, study, and implementation of two robot arm manipulators are described. This chapter also provides results on both robot arms, as well as a discussion of the final choice for actuation. Chapter 3 introduces the implementation of a robot arm network platform. This chapter goes through all of the parts of the platform

and provides several examples of its application on the physical robots. Results of the robot arm network are provided for validation. Chapter 4 show the experimental validation of the Cartesian coupled consensus algorithms developed [13, 14]. This chapter provides study on both controllers, both simulations, and experimental results for comparison. Chapter 5 then presents several possible applications that involve the combination of previous sections of the thesis. This chapter also provides a simulated example of one application of integrated use. Finally, Chapter 6 provides some conclusive study on the results as well as future work goals and progress.

Chapter 2

Single Robot Arm Development

2.1 Introduction

In this thesis, there was a goal to create a consensus based robotic arm network that can work alone or with the currently present AmigoBot research platform. In order to facilitate the creation of such a testbed platform, a suitable robot arm must first be found. An extensive study of robotic arms was performed in preparation of finding a robot arm to use in conjunction with the AmigoBot platform. The robot arm requirements are described in greater detail, however the robot arm must be suitable for any amount of research in consensus or in controls. The robot arm also needed to be light enough to be placed on the ground platform. Looking through off-the-shelf available robot arms, many robot arms were found that fit the requirements for design, but usually failed in another section of requirement. For this reason, it was decided that a robot arm needed to be built and tailored to the application. Putting together a robot arm would ensure that all the design requirements are met, and will allow for deeper understanding and adaptability if the project needs to change in future research pursuits. In this thesis, an open source solution was first pursued as a solution for a robot arm. In this chapter, the implementation, completion, and results of this project are described. However, construction time was deemed to be time exhaustive and during the research of this thesis a second option was discovered. For this option, custom off-the-shelf items were used to create a robot arm. The implementation, completion, and results of this arm are also presented within this chapter. Finally, a comparison and complete reasons for the decision change are included for clarity.

2.2 Design Requirements

To solve the problem of a distributed network of manipulators, a robot arm with a set

amount of features and possibilities must first be found. There are several goals that need to be fulfilled for a robot arm to work within a distributed network. To allow the robot arm to work within the research requirements and goals of the lab and of the distributed network platform, the robot arm must fulfill the following goals.

1. The robot arm must be light enough to be carried by the MobileRobots, Inc AmigoBot ground robots or other of similar size.
2. The robot arm must be cost effective, being cheaper than the mobile platform it will sit on.
3. The robot arm must be capable of delivering feedback on its state to induce a closed-loop control capability.
4. The arm must be easily reconfigurable and flexible to meet a research facilities current driven desires.

For the first constraint, the robot arm must be light enough to be carried by the current model of MobileRobots, Inc Amigobot that is used as a research tool within our lab. This constraint alone negates a large portion of off-the-shelf robot arms that are designed for industrial applications and are usually very much heavier than a small two wheeled medium sized robot will be able to handle. The robot arm must attach to the top of the on-board computer which sits on 2 inch standoffs on the robot. Because of this, the robot arm will be very high from the center of gravity of the robot. So even if the AmigoBot can hold the robot, the arm must still be light enough as to keep the robot from being too top heavy, possibly tipping over. Given such weight restrictions, limits the search of robot arms down to off-the-shelf robot arms that are able to be carried by our mobile platform. Within that requirement, the robot arm must then fulfill the other three constraints as found below. The approximate weight and size of the robot were not narrowed down, however the size of robot arms available off-the-shelf tended to either a very small size or extremely large. Final weight restraints were determined based on those off-the-shelf robot arms that were of small size and could fulfill the other constraints.

For the second constraint, the robot arm must be affordable enough to purchase in large quantities. This was a self imposed limit of choosing a robotic arm that is only as expensive as the ground robot. The ground robots come at a cost of approximately 3000 dollars. Therefore, a less expensive robot arm would allow for an affordable replacement due to common failures in experimentation. The robot arms could also be purchased in a large portions without becoming excessively expensive. This requirement is purely subjective, however it seemed reasonable as financial requirements should be weighed when searching for research parts and additions to the research lab.

For the third constraint, the robot arm must be capable of delivering feedback on its state to induce closed-loop control. This requirement is the most important for consensus based schemes and general control schemes. All of the lab's research is based on the study of some group closed-loop control, therefore each robot arm must be able to present state information that can be observed by outside sources. Most off-the-shelf robots that fit within the above constraints lack the capability of position feedback let alone velocity feedback. A general robot arm system has internal control loops which drive each joint to a given commanded angle with no feedback on angular position, velocity, load torques, etc., outside the internal loop. In a research environment, of which this manipulator is intended, this simply will not do. This constraint limited the resources of off-the-shelf components and robot arms down to a very select group. Of that group of robot arms, few fit the other two constraints already presented. A table of several robot arms explored can be found within Appendix C. Because of such varying constraints, the implementation of a customized robot arm was going to be the next step in exploration of finding a robot arm that will work. A robot arm that can present both angular and velocity feedback from each joint is the most desired.

Finally, the constraint over flexibility and reconfigurability is extremely important. Once again, in a research environment or in a general problem solving scenario, a robot arm that can be reconfigured to a specific situation or adapted to test new and different control theories is extremely desirable. In search of, or in implementation of, a robot arm,

it absolutely must be programmable to facilitate the current requirements or any others imposed at a later date.

2.3 Open Servo Arm Design

Through a search of off-the-shelf robot arms, I made the decision that the best arm to fit all of the proposed constraints was going to be a custom designed robot arm. In search of possible candidates, I discovered a open source radio servo project called OpenServo [15]. OpenServo is a open source project that has been designed and developed through both the engineering and radio controller hobbyist community. A normal radio controlled servo consists of a motor, potentiometer, and analog circuit. The analog circuit receives a pulse width modulated (PWM) signal as its control input. The width of the signal pulse stands for the desired position of the servo. Although there are varying standards for the frequency range of a standard servo, it usually has a middle range where the servo sits in the center of its full degree range. Above or below this range swings the servo arm left or right of its center servo angle. The standard analog circuit within that servo implements a closed-loop controller to move the motor and use the potentiometer as feedback to meet the desired angle input from the PWM signal. Although this works well in a hobbyist realm, the servo provides limited capability if used in a robot arm. There is no method to receive feedback from the radio servo or to break the internal control within the servo to implement externally. The OpenServo project has worked to develop a replacement digital circuit that can be put in place of the analog circuit that sits within the servo. The OpenServo board is a programmable micrcontroller that can accept outside input and also send internal input back out of the chip. The current version developed, allows for feedback in position, speed, voltage, and power of the servo. The motor servo can also be precisely controlled through both speed and position. While other market digital radio servos have the capability to precisely control the servo, they do not provide as much feedback. This extra inclusion of feedback not only fulfills the controller requirements, but facilitates many of the issues required in future flexibility. The circuit board on each servo communicates externally through a i2c interface which can be commonly used in conjunction

with a computer and a converter. Finally, each board can be addressable, which means the construction is simplified and all motors can be put on the same bus. Further discussion of the OpenServo hardware along with other hardware and developed software is expanded on within this section.

2.3.1 Mechanics

Once the OpenServo project was decided on as a source for actuation, the next step was to figure out the construction of the mechanics for the robot arm. To save time in construction and provide a more stable platform, pre-made parts were searched for in creating a robot arm. At this point, it was decided that a robot arm could be created using the Lynxmotion 4 degree of freedom kit [16]. This kit comes with laser cut plastic pieces that can connect to standard radio servos. The connections are made with screws and adhesive that can be removed or modified as needed for flexibility. In doing so, I could use the OpenServo boards and replace the circuitry of radio servos used on the Lynxmotion arm, turning a hobbyist servo into a research tool as seen in fig. 2.1. Upon construction of the robot arm mechanics, it was discovered that the robot arm, while sitting in its position on the AmigoBot, could not reach the ground. To fix this problem, the wrist plastic joint was replaced with Lego [17] pieces found in large supply in the lab. This turned the wrist joint into a very long joint that can reach the ground, but also could be adjusted to original size if needed. Although the addition of Legos left a lack of cosmetic appeal, the arm becomes more functional with the addition of an adaptable length joint.

2.3.2 Hardware

There were three hardware components in addition to the radio servos on this implementation of a robotic arm. The first and most crucial piece of hardware was the OpenServo microcontroller chip. The OpenServo controller chip, in fig. 2.2, was developed by the OpenServo community [15]. The schematics for it are openly available, but complete chips were also available for purchase [18]. The OpenServo board is configured from a Atmel 8-bit microcontroller along with H-bridge chips built out of MOSFET drivers along with

an EEprom to facilitate programs and servo register storage. The board is manufactured together to allow for power, motor, potentiometer, and communication connections to be attached. The chip can be talked to by using a standard i2c communication protocol. The board comes with the OpenServo software flashed on the EEprom, however a different or modified program can be loaded using a i2c bootloader and software such as AVRdude [19].

With the implementation of robot boards attached to the individual radio servos, there must then be a method of which to talk to the program available on the OpenServo board. For this thesis research, I decided that the best approach for flexibility is to leave the high-level control programs on a host computer and only use external hardware for low-level communication and translation. With this in mind, there needs to be a method of which a external host computer can talk to each OpenServo based servo on the i2c network. One solution determined was an open source interface board (OSIF) created by Barry Carter [18]. This board converts serial communication through a USB port to i2c communication line. This board, in fig. 2.3, was chosen as it was specifically developed for the OpenServo project.

The final piece of hardware needed is a board to power all of the OpenServo based servos and also to facilitate a i2c bus between each actuator and the OSIF board. This was simply created by soldering up a breadboard with headers for each motor cable and the i2c input from the OSIF. Additionally the power leads of each motor cable are connected to a circuit with a switch and 5-volt regulator that can be connected to any standard NiCad

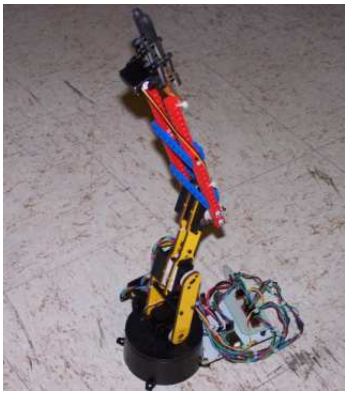


Fig. 2.1: Open servo robot arm platform.

Fig. 2.2: Open servo board.

rechargeable battery.

2.3.3 Embedded Software

The current open source software package for the OpenServo project is designed to talk to all the internal hardware components of the board and maintain a control table of registers that can be read or modified. For the purposes of this research, only a few registers are used. The open source program is designed to act exactly like a normal servo circuit using the desired angle registers as a desired angle for a internal software PID loop. The open source software additionally updates position and velocity registers with each iteration of the PID control loop. Each register for desired angle, desired velocity, current angle, and current velocity are set up in a high byte and low byte configuration. Therefore, any modification or access of these registers must be done so as to maintain the current byte configuration. A complete list of the control registers and board registers can be found in Appendix A. Once the software has initialized all the board components, the software enters a loop and calls three functions. The first function updates the state registers on the board by reading and calculating current values of the servo. The next function calculates a velocity for the motor using standard PID control theory, the values of the POSITION and VELOCITY registers, and the desired angle and velocity written within the SEEK_POSITION and SEEK_VELOCITY registers. Therefore, an outside program just needs to modify the SEEK_POSITION and SEEK_VELOCITY registers of the board through i2c and then the servo will internally control to that position at a given velocity. A general overview of the program can be seen in fig. 2.4.

In order to develop an arm which can be externally controlled, the elements of each

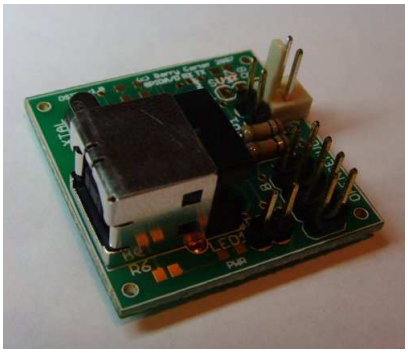


Fig. 2.3: OSIF i2c to USB board.

servo that facilitate the internal control of the servo were commented out in the embedded open source software. Instead, the PID loop was left only with two functions. The function to update the state registers for the servo was left in place. The function to write the velocity of the motor was modified to bound the servo inside a safe range within the mechanical limits of the servo. The servo used for the robot arm are the HS-311 Hitec servos. Their standard operating range is 180 degrees. To protect the motors from burning out in the event of a loss of communication from the external lines to the servo, the function that sends a velocity to the motor is modified to only send a velocity to the motor if the current register position values for the servo fall within its range of motion. Also, the velocity write function is modified to drive the motor based on the SEEK_VELOCITY register of the board instead of the velocity value from the PID function. After modification of the embedded software, the new modified software is then flashed onto the board using a i2c programmer cable and the software AVRdude [19]. With this configuration, an outside program then modifies the SEEK_VELOCITY register to drive the motor at a given velocity, but can still read the POSITION and VELOCITY registers for feedback. All of this still is performed using the i2c communication line. A general overview of the modified program can be seen in fig. 2.5.

Finally, as each board is flashed with its modified program, the modified program is also implemented with a given standard address configuration representing a shoulder, elbow, or wrist joint for later control from a host software program.

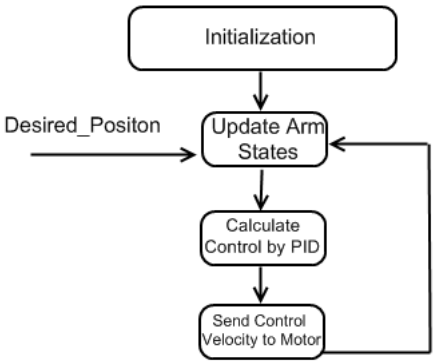


Fig. 2.4: Open servo standard program flow.

2.3.4 Host Software

To communicate with each OpenServo actuator, the (OSIF) board is used to facilitate translation of commands from a USB port to i2c protocol. Along with the board is the inclusion of a OSIF.dll library written by Barry Carter [18]. This library includes functions for initializing a bus of addressed motors and also functions for writing and reading from the register of each OpenServo board. Utilizing this library, a host software program was written with five functions that can be used to control a robot arm.

The first function written is used to set up the communication port with the OSIF and initialize all the servos and protocols. First the function attempts to connect with the OSIF board. If an OSIF board is present on the USB port then the function continues, otherwise it sends an error and quits. Next this function, calls another function to scan for the number of servos on the OSIF i2c bus. For this program, the function looks for the addressed servos that represent the shoulder, elbow, and wrist joints. If those joints are all present, then the function returns without error, otherwise it quits and presents an error. This function must be ran first before any other functions can be called to work with the OpenServo boards.

The second function written is to facilitate reading the registers for feedback of the servo. This function accepts an address for the motor and then calls library functions to see if that address exists on the i2c bus. If it does, the function continues otherwise it errors.

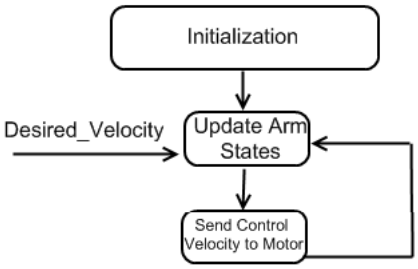


Fig. 2.5: Open servo modified program flow.

Once the motor is confirmed to exist, the function then calls another function to read a 6-byte region of the control table. The registers POSITION, VELOCITY, and POWER are read together. This control table information is then parsed into 2-byte sections representing current angle, velocity, and current load. These values are then passed by reference before the function ends.

The third function written is to facilitate writing the register for desired velocity of the servo. This function accepts an address for the motor and a desired angle for the motor. The function calls library functions to see if that address exists on the i2c bus. If it does, the function continues otherwise it errors. Once the motor is confirmed to exist, the function converts the desired angle into a 2-byte form and then writes to the SEEK_VELOCITY high and low byte registers. The function errors if the library write function returns an error.

The forth function written is to facilitate external PID control of a given OpenServo servo. This function accepts in a motor address and a desired position. The function starts out by writing a zero velocity to the addressed motor. The velocity is set up with the value 255 being a zero velocity. Below that value, 0-255, the motor turns at a rate counter clockwise. Above that value, 255-510, the motor turns at a rate clockwise. Next the function enters a conditional loop and reads the current position value of the motor utilizing the reading function already described. The function then calculates a velocity, given position history accumulated from each loop. The current position and current velocity are then

combined to calculate a PID derived control velocity. The control velocity is saturated to the 0-510 region of control. Finally, that value is written to the motor. The conditional loop continues until the current position is within a specified region of the desired position and then the function quits.

The final function written is to facilitate external PID control of two motors at once. This function was created to facilitate control of the shoulder joint of the Lynxmotion arm. The Lynxmotion arm uses two servos for the single joint. When trying to use the single PID control function to control two motors sequentially, it was discovered that each motor would have a slight delay and therefore the motors would fight each other. To prevent burn out of the shoulder joint, this function accepts two motor addresses and a desired position. It then doubles up on the reading, writing, and PID calculation. All of the functionality is the same as the single PID control function. This improvement of the function allows for the shoulder joint to run and the motors do not burn out.

The main program then sets up a menu for which each function can be called and address information can be entered in to test each function through the console. This provided general capability to study the robot arm and its ability. No further implementation of a program was created as this robot arm was abandoned for a better robot arm for reasons described later.

2.3.5 Results

To facilitate future consensus control of this robot arm within a network, the PID function was written to control each joint. If a OpenServo arm could be controlled externally by driving velocity and receiving feedback in angle and velocity, then the arm can be used for consensus purposes. To prove this, the wrist joint of the robot arm was tested by calling the PID function of the host software. The servo range operates in a value system from 0-1024 representing an degree range of 0-180 degrees. For this analysis, the wrist motor was asked to drive to 800 which represents 140 degrees. The result of this PID actuation can be found in fig. 2.6.

The program is able to drive the motor to the desired angle within 27 loop iterations.

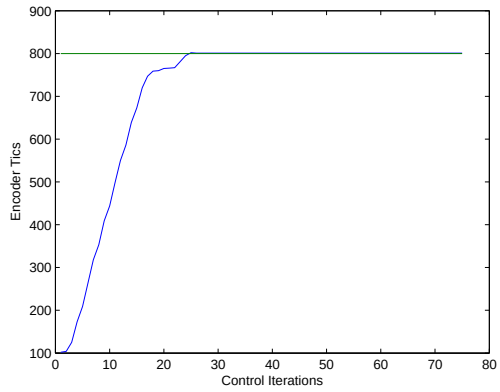


Fig. 2.6: Open servo PID controller result.

The speed and accuracy of the response can be modified using different PID gains, however for the purposes of the research, proving that the motor can be controlled externally proves that the arm is a good candidate for consensus control.

2.3.6 Change in Design

The Lynxmotion and OpenServo combination proves to be a great project for robotic arm research and for inclusion in a consensus based robotic arm network. However, a single robot arm requires extensive construction. Each motor cable must be custom made and each OpenServo board must be individually soldered and trimmed to fit into each individual servo. Finally, each arm must be mechanically built and a i2c bus must be created. This project would work well in the study of single robot control algorithms, however construction hampers its use in a large scale robotic network. Additionally, the OpenServo project is an open source project, therefore many of the issues inside the project board have still not been fixed. This means that occasionally the robot arm circumes to noise issues along the i2c network. This causes unpredictable results with the robot arm. Due to the fact that it takes two weeks to properly construct, and there are unpredictable natures to the arm, a better arm was desired that could be constructed faster and with less signal noise issues.

At his point, Vaibhav Ghadiok, a colleague within the lab, suggested looking into a platform robotics kit called Bioloid which uses Robotis Ax-12 motors. Upon further investi-

gation, these motors proved to be very close in development to the OpenServo project, but with most of the noise issues already worked out. Also, the kits came with microcontrollers to talk to a network of Ax-12 motors. Finally, each kit came with mechanics for hooking each motor up with other arms. The aspects of these kits forced a notable change to design a robot arm with these off-the-shelf components, while utilizing the wisdom gained in the Lynxmotion/OpenServo project arm in development of this new arm. The new arm is described in the following section.

2.4 Bioloid/Robotis Arm Design

A robot arm, created using the Robotis [20] AX-12 motors and the Bioloid [20] CM-5 microcontroller and mechanics, is used to facilitate the completion of a suitable research arm for inclusion in the research of a consensus based robotic arm network. The Bioloid kit comes with a given number of AX-12 motors, a CM-5 mircocontroller that connects to a host computer through serial and connects with the AX-12 motors, and enough mechanics to build a robot arm. This robot arm project has two program sections to it. The first program section exists on the CM-5 microcontroller. This program acts as a bridge program and translates serial communications from the PC host computer into serial communications to the AX-12 motor network that lies on a half duplex line. The second section of code lies on the PC host. This section of code contains the functions needed to control each independent AX-12 motor or all AX-12 motors together in the robot arm configuration. The Bioloid kit and pieces come to a price less than 500 dollars, which facilitates immediately one of the design constraints. The addition of programmability and many mechanical parts also facilitates additional constraints.

2.4.1 Mechanics

The Bioloid kit comes with the required mechanics to build a multitude of various robot creations. For this project, the goal was to create a robot arm. A total of five designs were considered, however, the final decision to create a very simplistic robot arm with one motor per joint was considered. In this design, only one AX-12 motor would be used for each

joint, which are for rotation, shoulder, elbow, wrist, and gripper actuation. Original designs allowed the robot arm to be about 24 inches long, with two motors to lift the arm via the shoulder joint. In this design, the two AX-12 motors used for the shoulder joint had issues as they were slightly out of mechanical sync. Because of this, even though the motors were both given the same command, one motor would be slightly behind the other making one of the motors heat up and shut down due to its resistance. The AX-12 motors are designed with heat sensors which protect the circuitry from overheating or the motor burning up. Although similar functions were considered as with the previous robot design, these motors proved strong enough to handle one motor per joint. Each Ax-12 motor has preventative programmed features to shut down if overloaded. To avoid the robot arm shutting down during demonstration, the final robot arm is approximately 14 inches long. This allows the arm only to barely reach the ground in front of the robot. The lightness of the robot arm allows the need for only one motor per joint. This means that the motor will be less likely to overheat and fail and it also simplifies the kinematics and weight issues of the robot arm. This also lightens the arm and allows it to be easily carried by the AmigoBot. The arm is shown in fig. 2.7.

2.4.2 Hardware

The hardware components for this arm are combined into two parts. The first part is the CM-5 microcontroller, in fig. 2.8. The CM-5 Microcontroller is based on an Atmel Atmega 128 microcontroller. The unit also has hardware to facilitate serial communications,

Fig. 2.7: Bioloid arm platform.

voltage regulation, and charging. There is also an EEprom on-board the unit for storing large programs that can run directly off the unit. Additionally, the unit has several buttons and LEDs on-board for direct user input and output. Finally, the unit has hardware to facilitate the half duplex communication needed to talk to a network of AX-12 Motors. The CM-5 also includes a rechargeable battery to support mobile implementation of the AX-12 motors and CM-5. The microcontroller arrives with software to flash on a program for the microcontroller through the given serial communications from a host computer.

The next hardware component is the Robotis Ax-12 motor, shown in fig. 2.9. The AX-12 motors were created by Robotis. They include inside a motor, gearing box, potentiometer, Atmel Atmega 128 microcontroller, and subsequent motor drivers and regulators. Each AX-12 motor has an on-board program that has been flashed. This controller can respond to a multitude of protocols and routines that have been written by and designed by the Robotis community. Communication to the motors is through a half duplex network allowing for only one cable to communicate both ways with each motor. This allows for each AX-12 motor to be daisy chained to additional motors or sensors which reduces the need for long wires and communication issues. The AX-12 motor gear box is designed for continuous rotation, although by default, the AX-12 program sets the motor to only run within a 300 degree range, as the potentiometer can only read within this span.

The program is set up to respond to a control table of registers similar to the OpenServo project. The registers denote the tolerances on the motor, its ability of torque, speed, etc. The registers also denote the motors ability for continuous or noncontinuous rotation. The registers also include safety values for when the motor will automatically shut itself down due to current loads or torque maximums. Each motor has its very own unique motor address which also subsides within the control table. All values within the control table can be modified to change the characteristics of the motor. For the robot arm, each motor is assigned a specific address denoting whether it is a shoulder, elbow, wrist, gripper, or base rotational motor. The application of these addresses is done through a software program that comes with the Bioloid kit. Although the default program for each AX-12 was left

in place, there is the ability to flash onto each motor an individual program. The default program of the Ax-12 motors is flexible enough to run strictly off the control table. Therefore by changing the values on the table, one can facilitate the behavior of the servo. The control table can be found in Appendix B.

2.4.3 Embedded Software

The CM-5 program was created as a bridge program to convert serial communication from a host PC computer into directions that can be executed on individual AX-12 motors connected to the unit. The original bridge program is an adaptation of the program created by Pedro Teodoro [21]. This program originally was developed to interface Matlab with one AX-12 motor through the CM-5. The program would accept a 2-byte packet from the host computer from Matlab. It would send the 2-byte packet denoting angle information to the AX-12 motor and then enter a loop sending back 2-byte packets of angle information read from the AX-12 motor to the host computer program running the Matlab simulation. This program is a modification of a open source general program provided as an example by Bioloid creators.

Using this program as a general guide, a bridge program was designed to handle three separate actions. The first action to handle would be writing an angle to the AX-12 motor to have the motor drive itself to a given 2-byte angle. At the start of the program, it enters a waiting loop and waits for a 2-byte packet to come through the serial line from the host computer. This packet denotes the opcode for one of the three actions to be performed.

If the opcode is 1, the program will enter the write angle function. This function waits for an additional 2-byte packet to come through denoting the motor address. The function

Fig. 2.8: CM-5 microcontroller.

Fig. 2.9: AX-12 Robotis motor.

then writes to the continuous registers of the appropriate motor so that the motor will only respond to angle information and will drive that motor to the appropriate angle at a given speed if provided, else a default is used. After this address is given, the motor then waits for a packet denoting the angle. After all of these transmissions are received, the function sets the specific AX-12 address register with the appropriate high byte and low byte angle information. The AX-12 motor then takes care of the rest.

If the opcode is 2, the program will enter the write speed function. This function waits for an additional 2-byte packet to come through denoting the motor address. The function then writes to the continuous registers of the appropriate motor so that the motor will rotate continuously regardless of the angle given and will only respond to motor speed. After this address is given, the motor then waits for a packet denoting the motor speed. After all transmissions, are received, the function writes to the high and low byte motor speed registers. The motor will rotate at the given speed in it's register.

If the opcode is 3, the program will enter the read angle function. This function waits for an additional 2-byte packet to come through denoting the motor address. The function then reads from that specific motors control table the high byte and low byte current angle. The values are combined and then the data is transmitted back to the host computer.

2.4.4 Host Software

The host software sits on a given host computer that is connected through serial to the CM-5 microcontroller and bridge program. The host software includes five functions which facilitate various capabilities of the robot arm. The first function deals with writing an angle to each motor. This function exists on the host computer program and is designed

to cause the particular addressed AX-12 motor to drive to a specific angle. The function starts up establishing a connection with the serial port of the computer. The function takes in two parameters. The function needs the motor address and the angle to drive to. The angle measurements run from 0 to 300 degrees, but the motor takes finite increments from 0 to 1024 in value. After the function establishes a serial connection, it sends the opcode of 1 to tell the bridge program its intention. The address of the motor is then sent over serial in a high and low byte configuration. The angle is also sent subsequently using the same configuration. Finally the serial port is closed.

The next function is designed to write a speed to an addressed motor. This function exists on the host computer program and is designed to cause the particular addressed AX-12 motor to drive to a specific speed. The function starts up establishing a connection with the serial port of the computer. The function takes in two parameters. The function needs the motor address and the speed to drive to. The AX-12 speed goes from 0-1024 increments for counter clockwise rotation and 1024-2048 for clockwise rotation. After the function establishes a serial connection, it sends the opcode of 2 to tell the bridge program it's intention. The address of the motor is then sent over serial in a high and low byte configuration. The speed is also sent subsequently using the same configuration. Finally, the serial port is closed.

The third function is designed to read state information from an addressed motor. This function exists on the host computer program and is designed to read the particular addressed AX-12 motor's current angle. The function starts up establishing a connection with the serial port of the computer. The function takes in one parameter. The function needs the motor address. The AX-12 angle goes from 0 to 300 degrees or 0-1024 increments. After the function establishes a serial connection, it sends the opcode of 3 to tell the bridge program its intention. The address of the motor is then sent over serial in a high and low byte configuration. The angle is then sent back to the program subsequently using the same configuration. The function reads the high and low byte angle and combines it into its increment angle translation. Finally, the serial port is closed and the angle is returned.

The forth function is implemented to externally control the addressed motor through PID control. This function allows the AX-12 motor to be externally controlled for cooperative algorithms and external control theory. This function takes in four parameters. The address of the motor, the angle to control to, and a high and low degree range to limit the motor to. This function utilizes the write speed and read angle functions along with the standard PID equations within a loop. For each iteration, the current angle is read and ran through the PID equations. The PID velocity is estimated using position error accumulated over change in time. Finally, a subsequent speed is written to the motor. The loop stops when the motor angle is achieved, the motor is then commanded to stop.

The final function is implemented to drive the robot end effector arm to a given spacial x,y,z position in reach of the robot arm. This function controls the specific motors of the Bioloid robotic arm to allow for the control of the position of the gripper at the end of the robot arm. The function takes in three parameters. The x, y, and z position in inches within the robot arm's spatial range. The program utilizes the write angles function to drive each motor to its given angle once those angles are calculated using the inverse kinematics equations. To calculate the inverse kinematics, the robot arm joint lengths must be known. The lengths of the arm are given by

$$l_1 = 4.5,\, l_2 = 4.5,\, and\, l_3 = 4.5, \tag{2.1}$$

where l_1 represents the shoulder, l_2 represents the elbow, and l_3 the wrist.

The inverse kinematics equations were designed to only allow for one method of obtaining the inverse kinematics position [22]. The equations calculate the require angle for the rotation, shoulder, elbow, and wrist motors required to drive the gripper to the required position. The equations used are as

$$q = \sqrt{1/(b_1^2 + b_2^2) - 1} \tag{2.2}$$

$$\theta_1 = \arctan\left(\left(\frac{z_c}{2l_2} + q\frac{x_c - l_3}{2l_2}\right), \left(\frac{x_c - l_3}{2l_2} - q\frac{z_c}{2l_2}\right)\right) = Shoulder\, Rotation,$$

$$\theta_2 = \arctan\left(\left(\frac{z_c}{2l_2} - q\frac{x_c - l_3}{2l_2}\right), \left(\frac{x_c - l_3}{2l_2} + q\frac{z_c}{2l_2}\right)\right) - \theta_1 = Elbow\,Rotation,$$

$$\theta_3 = -\arctan\left(\left(\frac{z_c}{2l_2} - q\frac{x_c - l_3}{2l_2}\right), \left(\frac{x_c - l_3}{2l_2} + q\frac{z_c}{2l_2}\right)\right) = Wrist\,Rotation,$$

$$\theta_4 = \arctan\left(Y_c, X_c\right) = Arm\,Rotation,$$

where values x_c, y_c, and z_c represent the coordinates desired for the end effector. Additional code is provided to prevent the robot arm from going out of bounds or the motor joints falling out of regions of operation. Conditions of the robot arm keep the individual motors saturated at their outer limit regions of actuation. This does introduce some error in the final effector's position, but does not cause an issue in this project. Finally, the x, y, and z parameters sent into to the function are checked to verify they are within the region of the robot arms capability. If they are not, an error is returned.

The robot program utilizes serial communication libraries created by Thierry Schneider [23]. These libraries were chosen for their simplicity and speed in communicating with the serial port. This provided more efficiency in achieving a robotic arm project quicker, since the goal is ultimately to facilitate a consensus based robot arm network.

2.4.5 Results

To facilitate future consensus control of this robot arm within a network, the PID function was written to control each joint. If a Ax-12 motor could be controlled externally by driving velocity and receiving feedback in angle and velocity, then the arm built of these motors can be used for consensus purposes. To prove this, the wrist joint of the robot arm was tested by calling the PID function of the host software. The servo range operates in a value system from 0-1024 representing a degree range of 0-300 degrees. For this analysis, the wrist motor was asked to drive to varying degree values from 65-175 degrees. The result of this PID actuation can be found in fig. 2.10.

The program is able to drive the motor to the desired angle within roughly 20-loop iterations each time. For the testing of the PID response, the motor response is not very accurate, however this was purely due to tuning the PID gains. The speed and accuracy

of the response can be modified using different PID gains, however for the purposes of the research, proving that the motor can be controlled externally proves that the arm is a good candidate for consensus control.

2.5 Implementation and Usage

The robot arm ended up being a bit more challenging then I first predicted. However, the testing went rather well for it. I learned real quickly that the robot arm motors cannot withstand a huge amount of torque created by a lengthy arm. Therefore, the arm was shortened to a length where it could still reach the ground, but it would be light enough for all of the AX-12 motors to handle. Originally, two AX-12 motors were going to be used in sync as at the shoulder joint. However, even with the same address the physical properties of the motors meant that they were just slightly off of sync and fought each other while running. This extra tension lead to many overheats and overcharge shutdowns of the motor, which would not work well in the demonstration. To alleviate this, a design decision to use one motor per joint proved useful, and the inverse kinematic algorithm worked nicely after this. The robot arm software can facilitate all that is required to develop a basic network of robotic arms for consensus research, therefore this arm was replicated in bulk to allow for research in implementation of a robotic arm network that is discussed in the next chapter.

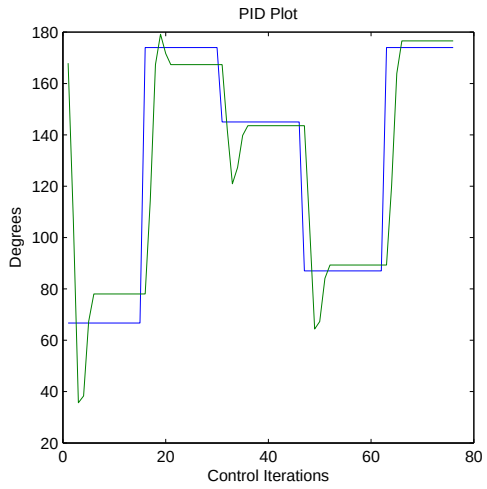


Fig. 2.10: Bioloid PID controller result.

Chapter 3

Robot Arm Network Testbed

3.1 Introduction and Motivation

In this thesis, there was a goal to create a consensus based robotic arm network, fig. 3.1, that can work alone or with the currently present AmigoBot research platform. While a group of ground robots are useful in solving problems by themselves, the addition of robot arm manipulators on each ground robot allows the robots to conquer even more complex problems and issues. As there has been a great deal of study with consensus of ground robots, there is little study with consensus based algorithms for robot arm manipulators. For this thesis, I designed a platform for research into consensus algorithms for a network of a given number of robot arms. There are several benefits to consensus based control of a robot arm network. Perhaps the robot arms need to carry an object across a room. Using consensus, one can allow the arms to synchronize with little computation power per robot arm and with only suggestive reference angles to one or a few of the robot arms. Similarly, cameras can be added to end effectors of the robot arm. Consensus can allow for the cameras to converge to the same angles allowing for group filming of an object by multiple agents. In fighting a fire, robot arm actuators can offset in convergence to opposite angles, therefore by cooperation of meeting angles, part of the group can fight the lower portion of the fire while the other fights the upper, all of which could be done through synchronization of the arms. The research provides varying interests and capabilities, and thus a platform testbed is desired to further pursue the possibilities using consensus control algorithms. Utilizing the Bioloid arm platform described and implemented in Chapter 2, the following presents the creation of a robot arm network testbed. Additionally, the robot arm network is tested through experimental validation to prove that it works.

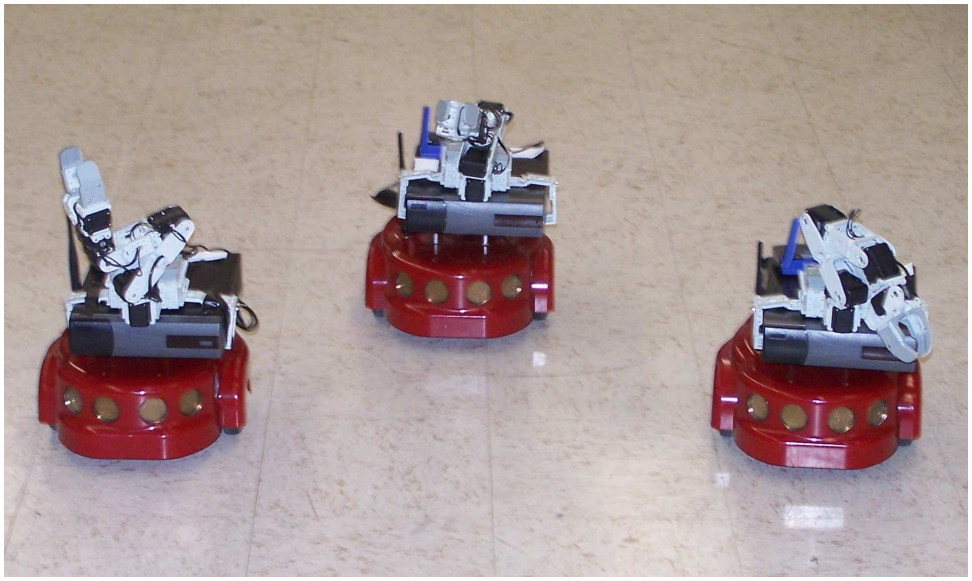


Fig. 3.1: Robot arm network platform.

3.2 Design Requirements and Problem Statement

To create a system of robot arm manipulators, assuming that they have met the single manipulator constraints, the system will have to meet a larger set of requirements that are needed for the purposes of experimental validation and research testing.

1. Each robot ground vehicle and the individual arm must be able to share commands and state information.

2. Each robot arm must have the capability to share state information with other robot arms.

3. The lines of communication must be alterable to facilitate communication topology testing between arms.

4. The system must be able to utilize a reference that one arm or all can follow if desired.

The first goal is to make a robot arm manipulator so that it can communicate with the ground robot and with other robots. In Chapter 2, the Bioloid robot arm can talk with any computer supporting standard serial communication. This constraint is immediately met as long as the connecting computer is on a local network with computers using other

Bioloid arms. Thus the addition of network software will be needed to get this constraint met at a program level.

The second goal is to allow each arm to share state information with other arms. This too becomes easier if the first requirement is met. Sharing information then becomes a task of sharing packets over the wired or wireless network.

The third goal is to allow the communication between arms to be altered as requested. In a perfect world, communication would never fail, but for the requirements of research, the ability to induce virtual comm failures proves useful in creating more robust and capable control schemes.

The final goal is for the system to utilize a virtual reference. This is very important, as the robot arms may not be just tasked to synchronize to each other, but also to a goal location or point. Because the system is decentralized, this requires the ability to assign a leader who can see the reference or the ability to control for experimentation purposes, which arms can or cannot see the reference.

3.2.1 High-Level Development

The platform for control is thus set up into two programs. The first program, fig. 3.2, is the reference program. The reference program has two modules. The first module is a server module which provides reference angles for shoulder, elbow, and wrist to any client program from any networked computer that asks. The second module is a client program which queries each robot arm within the system to gain state information for the purposes of logging the trajectories of the system of robot arms.

The second program, fig. 3.3, will be the program that runs on each robot arm computer. This program is broken up into two modules. The first module is a server module which provides current angle information on shoulder, elbow, and wrist to any client program from any networked computer that asks. The second module contains the consensus algorithm along with the client to obtain the given values from the other robot arms and then drive its robot arm using the program functions described in Chapter 2.

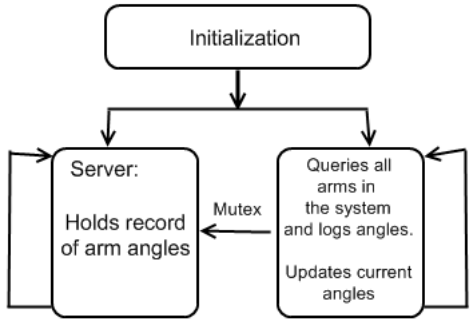


Fig. 3.2: Reference program flow structure.

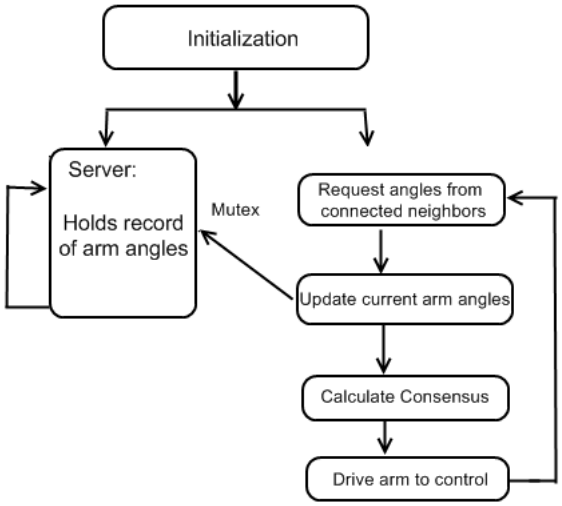


Fig. 3.3: Controller network arm program flow structure.

3.2.2 Network Communication

To allow each robot arm to share and get angle data from other robot arms within the network, a set of functions needed to be created to facilitate the sharing of data across a network. The easiest form of communication, would be to use the TCP transfer protocol across the network. For this project, a library of network functions was utilized. These functions contain TCP server and client code which are expansions of standard windows network protocol API functions. The library called Practical Sockets, is written by Anthony Prior [24]. Due to the fact that each program must run several modules at once. The

Pthread libraries are used to run simultaneous threads of execution. For the server module described above, it is put into a parallel thread with the main program.

Inside the server thread, the code locks a mutex and reads in the angles from the mutex. Once that information is read, the information is stored in the server thread values. The server than waits until a client program connects to its port number. The port number for the server of each robot arm is the value 677867 plus the number of the robot. So robot number one holds the value 677868. When a client program connects, the TCP server sends back the three angles parsed together in a single character string. The server spawn of its own threads which can individually handle client requests. This allows for multiple clients to connect to the server at once for information.

The second module used in network communication is the client module. This module is run upon request by the program. This module requires the IP address for the networked computer that it will request and the port number as described above. With this information, the client program sends a packet request to the server program. The server program sends back a character string containing angle values of that robot. These angle values are then parsed and converted back into data that can be used within consensus.

3.2.3 Control Structure

The second module of the consensus program for each robot runs in the main thread. This module runs parallel to the server module. This module first starts out by initializing the server thread and anything that the Bioloid arm needs to have. Communication topologies are also set up for each of the robot arms within the network. If the robot arm number is an arm that the current program can get state information values from, the value for that topology is a 1 otherwise it is a 0. This ability allows for research into the study of consensus in a network that is not completely connected. This is also designed so that the robot arm program can utilize the reference or not. Based on its ability to see the reference, it becomes a leader or a follower. The particular robot arm program is also labeled with a number which determines its IP address on the network and the port at which other robots can gain information from it.

Once the robot arm is initialized, it is sent its initial angular values. Within our ground robot platform testbed, the robot positions are put in place, and gravity holds them in place. However, with a robot arm, the initial positions of the robot must be fixed on the robot. To facilitate this, the write angle function is used to lock the robot arm in its given initial positions before the main consensus algorithm runs. Once the algorithm is run, the positions of the robot are free to change, but to hold the arm in a given position, the function to internally control the arm to an angle is used.

The robot arm program then enters a loop where the consensus algorithm is performed. The functions to obtain the current angles of the robot arm are first called. The corresponding values are then placed into a mutex that is shared with the server module. As all of the robot arms may start at varying times within the system, the loop will continue only to update its state information for a controllable amount of time. This ensures that no consensus will start until all of the robot arms are ready at once.

Once the time delay has completed within the loop, the consensus algorithm is activated. Within this algorithm, the client modules get angle data for each robot arm that the particular arm program can talk to. The algorithm then calculates the required velocity for each joint using the standard consensus algorithm as

$$u_i = -\sum_{j=1}^{n} a_{ij}(r_i - r_j),\ i = 1, ..., n, \tag{3.1}$$

where u_i represent the control input, r_i is the state of the ith robot arm, r_j is the state of the jth neighbor, $a_{ij} > 0$ if the ith robot arm can receive information from the jth one, and $a_{ij} = 0$ otherwise. The adjacency matrix is a matrix representation of the communication topology graph describing whether or not agents in a system can communicate with each other. Once a control input is determined, the value is checked to make sure it falls within the saturation requirements of the motor velocity as described in Chapter 2. The control loop continues until the program is ended.

3.3 Experiments and Results

To test the capability of this robot arm network platform, two communication topologies were tested with three robot arms. The first topology test was with a completely connected topology and one reference host program. The reference host program for the first test has shoulder, elbow, and wrist angles found as

$$\begin{aligned} Shoulder\ \theta &= 88\ degrees, \\ Elbow\ \theta &= 132\ degrees, \\ Wrist\ \theta &= 50\ degrees. \end{aligned} \tag{3.2}$$

The initial values for each numbered robot arm can be found as

$$\begin{aligned} Shoulder\ \theta_1 &= 203\ degrees, \\ Shoulder\ \theta_4 &= 87\ degrees, \\ Shoulder\ \theta_5 &= 86\ degrees; \end{aligned} \tag{3.3}$$

$$\begin{aligned} Elbow\ \theta_1 &= 88\ degrees, \\ Elbow\ \theta_4 &= 40\ degrees, \\ Elbow\ \theta_5 &= 41\ degrees; \end{aligned} \tag{3.4}$$

$$\begin{aligned} Wrist\ \theta_1 &= 147\ degrees, \\ Wrist\ \theta_4 &= 60\ degrees, \\ Wrist\ \theta 5_1 &= 49\ degrees; \end{aligned} \tag{3.5}$$

where the index number references the AmigoBot on-board computer the arm is attached to. The communication topology for the first test is a completely connected topology. This means that every robot arm can see every other robot arm and all robot arms can see the reference. Figure 3.4 shows a graph to represent the topology. Figures 3.5, 3.6, and 3.7 show the results of the experiment.

In the results, one can notice that the angles for each arm stay at given angles for many control iterations. In reality, the arms move at varying angles in between, but the

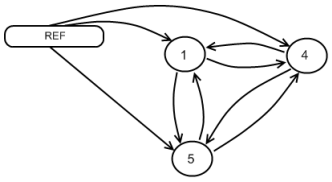


Fig. 3.4: Full connected topology.

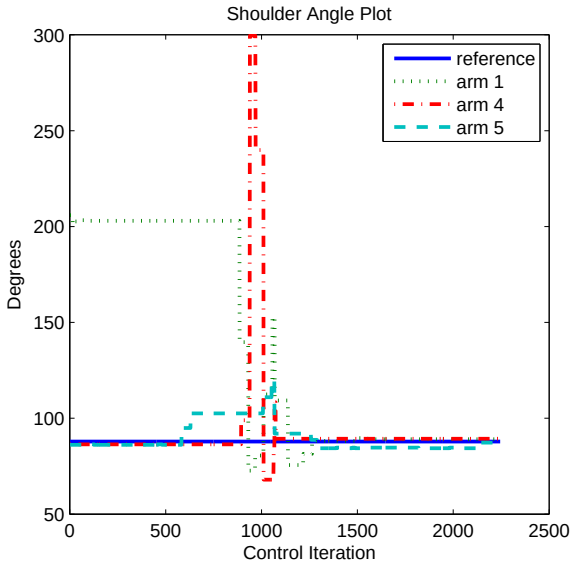


Fig. 3.5: Shoulder angle consensus, fully connected.

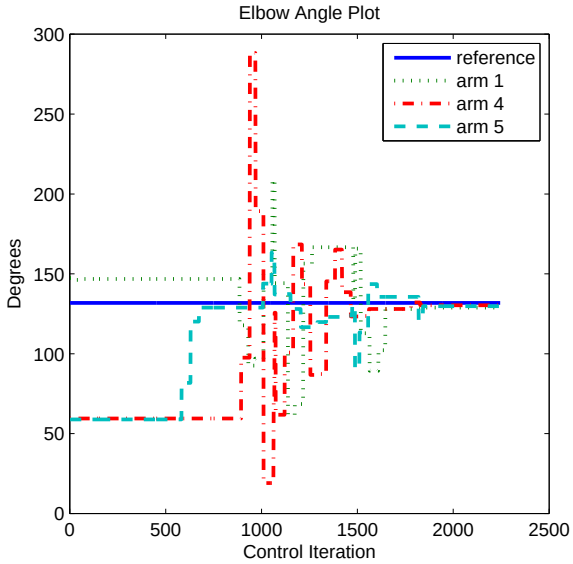


Fig. 3.6: Elbow angle consensus, fully connected.

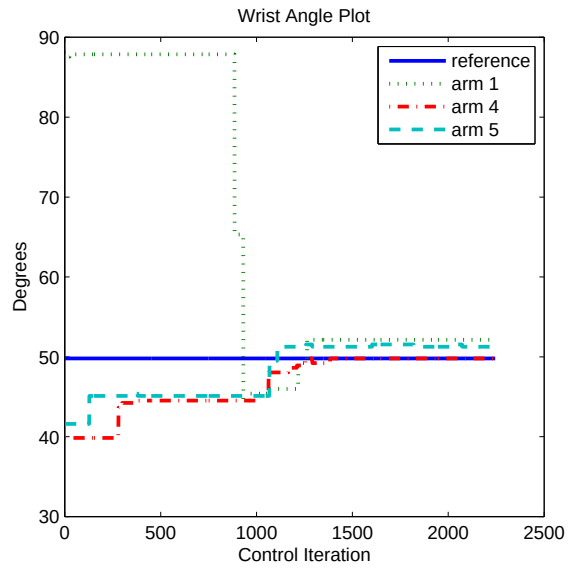


Fig. 3.7: Wrist angle consensus, fully connected.

data is sampled from the reference program which samples at the lowest priority to the other robots within the network, to facilitate small delays in the consensus algorithm. However, it is shown that the robot arms within the network converge to the reference signal and to each other. In a completely connected topology, on average, the error between arm convergence is 2-3 degrees.

The communication topology for the second test is a partially connected topology. This means that every robot arm may not see other robot arms or the reference. However, the topology graph must be at least a direct spanning tree to allow for consensus to be stable. The reference host program for the second test has shoulder, elbow, and wrist angles found as

$$\begin{aligned} Shoulder\ \theta &= 80\ degrees, \\ Elbow\ \theta &= 137\ degrees, \\ Wrist\ \theta &= 59\ degrees. \end{aligned} \tag{3.6}$$

The initial values for each numbered robot arm can be found as

$$\begin{aligned} Shoulder\ \theta_1 &= 203\ degrees, \\ Shoulder\ \theta_4 &= 86\ degrees, \\ Shoulder\ \theta_5 &= 120\ degrees; \end{aligned} \tag{3.7}$$

$$\begin{aligned} Elbow\ \theta_1 &= 147\ degrees, \\ Elbow\ \theta_4 &= 85\ degrees, \\ Elbow\ \theta_5 &= 171\ degrees; \end{aligned} \tag{3.8}$$

$$\begin{aligned} Wrist\ \theta_1 &= 88\ degrees, \\ Wrist\ \theta_4 &= 44\ degrees, \\ Wrist\ \theta 5_1 &= 131\ degrees; \end{aligned} \tag{3.9}$$

where the index number references the AmigoBot on-board computer the arm is attached to. Figure 3.8 shows a graph to represent the topology used in this test. Figures 3.9, 3.10, and 3.11 show the results of the experiment.

In the results, one can notice that once again the angles for each arm stay at given angles for many control iterations. In reality, the arms move at varying angles in between, but the data is sampled from the reference program which samples at the lowest priority to the other robots within the network, to facilitate small delays in the consensus algorithm. However, it is shown that the robot arms within the network converge to the reference signal and to each other. In this particular connected topology, on average, the error between arm convergence is slightly larger at 6-7 degrees maximum.

3.4 Conclusion

The robot arm network platform proves to work to utilize consensus together with a robot arm network. The convergence work well, however there can be more improvement in the future to optimize delays between robots. Additionally, this platform will not work for larger systems without the addition of semaphores to the current running thread mutex. Additional capability to the communication topology and the robot program could be added

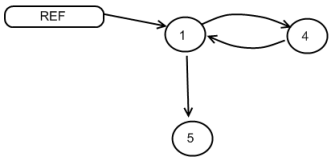


Fig. 3.8: Partially connected topology.

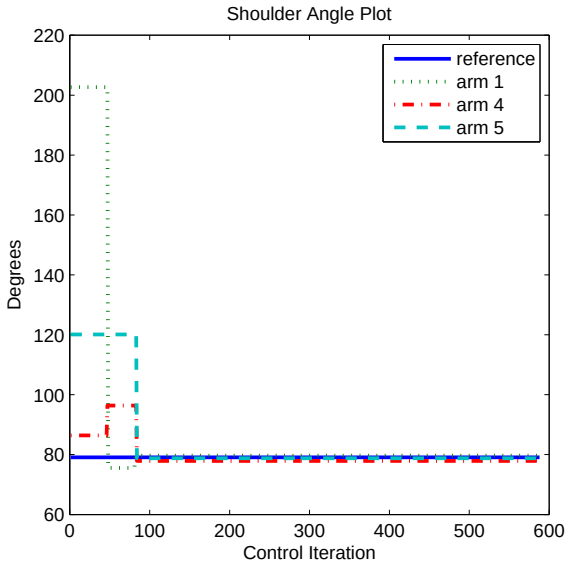


Fig. 3.9: Shoulder angle consensus, partially connected.

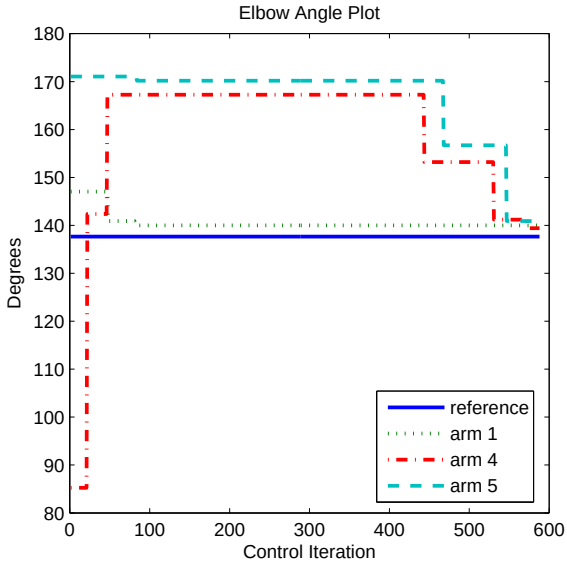


Fig. 3.10: Elbow angle consensus, partially connected.

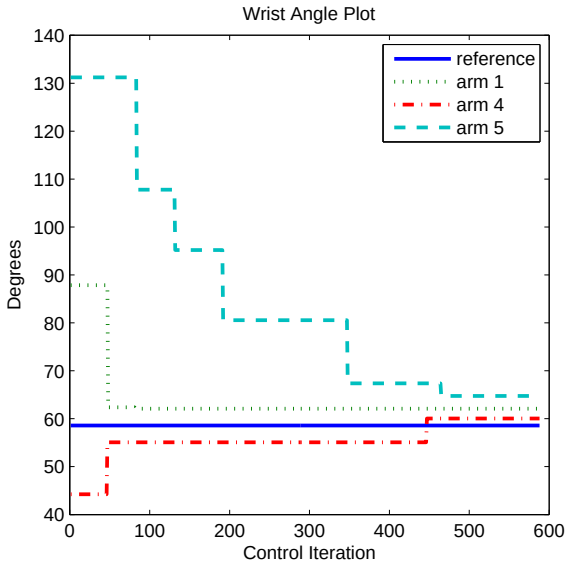


Fig. 3.11: Wrist angle consensus, partially connected.

for even more research capability. Finally, a program could be written to initialize and signal all of the robots in the system for more efficient experimentation of the robot arms where all arms are ran at once and consensuses at once without the need for a user to start up each program individually. Overall, the platform satisfies the set forth requirement of the system and creates a basic testbed for consensus research through robot arms on static or mobile platforms.

Chapter 4

Cartesian Coordinate Coupled Control

4.1 Introduction and Motivation

Current research theory within the lab has surrounded around the idea of consensus [7]. The concept of consensus is the idea of allowing each robot within a system to negotiate its position in contrast to the position of the other robots it can talk to within the system. In such a concept, instead of a group of robots being commanded by a central source, each robot within the system negotiates its state with other robots within the system, i.e, they come to a consensus on where to meet. This can best be described as synchronization or a weighted average of states. For example, one robot may have access to the reference position, and therefore it becomes a leader within the system. On the other hand, all of the robots may only have access to other robots within the system, therefore being decentralized in control. For robots with kinematics $\dot{r}_i = u_i$, an equation for control can be found as

$$u_i = -\sum_{j=1}^{n} a_{ij}(r_i - r_j),\ i = 1, ..., n, \tag{4.1}$$

where r_i is the state of the robot, r_j is the state of the jth neighbor, u_i is the control input for the robot, and a_{ij} is the (i,j)th entry of an adjacency matrix, as described in Chapter 3. In this matrix the (i,j)th element represents the communication flow from the jth to the ith robot. If the value is 0 then communication does not occur. If (i,j) is larger than 0 then the ith robot can obtain information from the jth robot neighbor. For the purposes of consensus, if the corresponding graph for a adjacency matrix, as in fig. 4.1, has a directed spanning tree, the consensus is guaranteed to be achieved [7].

Finally within consensus, the control command input u_i represents a single state variable. For example, in a ground vehicle based within the standard consensus of (4.1), the

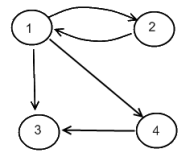


Fig. 4.1: Partially connected topology.

x coordinate is consented together and then the y coordinate is consented together. In general, the Cartesian coordinates within a general consensus equation remain decoupled.

Understanding the features of standard consensus, we can now move on to collective motion and Cartesian coupling. Within the current consensus based control theory, many of the control schemes have been validated to be true on physical discrete time-based mobile robots [12]. Control theory and simulation over various forms of consensus for both single integrator kinematic and double integrator dynamic Cartesian coupling [13, 14]. However, the validation of current theories of combining leaderless collective motion and Cartesian coupling have not been tested on physical discrete time mobile platforms.

Synchronization and collective motion can be found all around nature. Large galaxies follow orbits and logarithmic spirals very similar to collective motion. Groups of animals from fish to birds to stampedes all move as a group using some form of collective motion. Groups rely on their interdependent positions to move individually causing a collective motion of the group. In robotics research, the use of collective motion has huge benefits in groups of robots solving problems such as surveillance, search and rescue, and patrolling functions.

Cartesian coordinate coupling within consensus is very similar to general consensus with the addition of a rotation matrix. A general form of coupled Cartesian coordinate consensus is given by

$$u_i = -\sum_{j=1}^{n} a_{ij} C(r_i - r_j),\ i = 1, ..., n, \tag{4.2}$$

where all remains the same to the standard consensus equation, but with the addition of the rotation matrix C. For this thesis, the matrix C is a rotation matrix, however it could

be any general matrix. The addition of the C matrix allows for coupling of coordinates between the robots. For example, if I am calculating the x coordinate state for my ground robot, the velocity in x is now calculated as a weighted average of both the x coordinate and the y coordinate of neighbors. To better understand this, we can convert the general consensus equation into the general form (4.3). This equation is now the Laplacian matrix $\mathcal{L}$ along with the Kronecker product of an identity matrix. The Laplacian matrix is the matrix representation of the directed spanning tree forming the communication topology of the multi-agent system. The combination is multiplied with r where $r = [r_1^T, ..., r_n^T]^T$. The general form equation for consensus is given as

$$\dot{r} = -\left(\mathcal{L} \otimes \mathcal{I}\right) r, \tag{4.3}$$

where $\mathcal{I}$ is the identity matrix and $\mathcal{L}$ is the Laplacian matrix. In the Laplacian form of this equation, stability of the system is based on the stability derived from the eigenvalues of the Laplacian matrix. If the graph interaction has a direct spanning tree, consensus is achieved. Now we can look at this equation modified to be Cartesian coordinate coupled as

$$\dot{r} = -\left(\mathcal{L} \otimes \mathcal{C}\right) r, \tag{4.4}$$

where the Kronecker product of the Laplacian $\mathcal{L}$ is now with the rotation matrix C. Now, if stability is analyzed the eigenvalues of both the Laplacian and the rotation matrix C must be analyzed. Based on the properties of the Kronecker product, the eigenvalues to be analyzed for stability are the eigenvalues of the Laplacian rotated by the angle θ that is used within the rotation matrix. By varying the rotation C from θ larger than 0. The eigenvalues will be rotated outwards towards the imaginary axis. At some point θ, the eigenvalues are pushed onto the imaginary axis. This provides for the system to be marginally stable. Further increase of the θ pushes the eigenvalues into the right hand side bringing on instability. Within this controller, a small θ causes all vehicles to cyclically pursue each other at a set angle apart logarithmic spiralling into convergence. At a critical θ, the vehicles follow

circular orbits at given angles. Finally, larger θ values present logarithmic spirals that spiral out and eventually become unstable [13, 14].

In general consensus, a group of UAV surveillance aircraft can only converge together or fly in formations. With the introduction of Cartesian coupling, convergence turns into logarithmic spirals inwards, divergence becomes spirals outward, and marginal stability will cause orbiting pursuant aircraft. In Cartesian coordinate coupled collective motion, the states of the trajectory of each robot (x,y,z) are coupled together in a consensus control through the use of a rotation matrix. The use of a rotation matrix in conjunction with general consensus algorithms creates the ability to control the motion and stability of a system through the control of the rotation angle of the rotation matrix. By varying a single angle, the system of robots cyclically pursue each other while converging in a stable manner, diverging in an ultimately unstable manner, or orbiting in a marginally stable manner. For a group of robots searching or covering a large area, the ability to spiral in and out or orbit allows for better coverage of an area, and less possibility of missing something. This ability becomes greatly useful in anything from cleaning carpets, patrolling outward in an aerial search pattern, or spraying down a fire from the outside in. In this thesis, two Cartesian coordinate coupling controllers are studied. One controller is derived for single integrator kinematics and the other for double integrator dynamics. Both controllers are studied in both computer simulation and on a physical robot platform system.

4.2 Design Requirements and Problem Statement

To run a platform for testing the consensus algorithms the AmigoBot robots must follow a software platform with the following constraints.

1. Each robot ground vehicle must be able to share commands and state information.

2. The lines of communication must be alterable to facilitate communication topology testing between ground robots.

3. The system must be able to except a reference that one ground robot or all can follow if desired.

The first goal is to share state information and commands. The current consensus structure is set up to share position data between ground robots using a tcp server/client relationship. Therefore, this requirement is already met due to previous research within the lab.

The second goal is to allow alterable communication lines to facilitate topology testing. Within the current AmigoBot testbed, a topology adjacency matrix is shared between all robot agents through a central system. Therefore, this requirement is also met.

The final goal is for the system to except a virtual reference. This is very important, as the ground robots may not be just tasked to synchronize to each other, but also to a goal location or point. Because the system is decentralized, this requires the ability to assign a leader who can see the reference or the ability to control for experimentation purposes, which ground robots can or cannot see the reference. This requirement is also met in the current AmigoBot testbed.

As all requirements are met, the experimental validation will only require adjustments and adaptations of current control algorithms to get the controller to work in the present consensus based AmigoBot testbed.

4.3 Controller Development

4.3.1 Single Integrator Kinematics

The first controller is used for a vehicle which uses single integrator kinematic given by

$$\dot{r}_i = u_i,\ i = 1, ..., n, \tag{4.5}$$

where $r_i = [x_i, y_i]^T$ is position and $u_i = [u_{x_i}, u_{y_i}]^T$ is a control input. The equation that will be used for this controller [13] is given as

$$u_i = -\sum_{j=1}^{n} a_{ij} C(r_i - r_j),\ i = 1, ..., n, \tag{4.6}$$

where $C = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$ is the 2 x 2 rotation matrix, and θ is the rotation angle. The single integrator controller kinematic controller is best used for vehicles such as the lab AmigoBot that requires a control input of velocity. This thesis will provide both simulation and experimental results in the following sections.

4.3.2 Double Integrator Dynamics

The second controller is used for a vehicle which uses double integrator kinematic given by

$$\dot{r}_i = v_i,\ \dot{v}_i = u_i,\ i = 1, ..., n, \tag{4.7}$$

where r_i is position, v_i is velocity, and u_i is a control input. The equation that will be used for this controller [14] can be found as

$$u_i = -\sum_{j=1}^{n} a_{ij} C(r_i - r_j) - \alpha(v_i - v_j),\ i = 1, ..., n, \tag{4.8}$$

where $C = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$ is a rotation matrix of two dimensions for the x and y coordinate of a ground vehicle, which provides the coordinate coupling. The value α is the gain on the velocity error within the system. Together this controller forms a relative damping Cartesian coordinate controller. The double integrator controller dynamics controller is best used for a wide variety of vehicles found in the robotics industry which are precisely controlled through motor torques or acceleration. This thesis will provide both simulation and experimental results in the following sections.

4.4 AmigoBot Platform

The physical robot experiments are based on four AmigoBots from ActivMedia Robotics. All four robots are driven via differential drive. They have high precision wheel encoders

and eight sonar attached around the robot. Within each robot is a 16-bit micro-controller that maintains support for all sensor and actuation features of the robot. A control program platform is then run remotely over a WAN. The host computer executes programs which send and receive sensor data and control actuation to the robots on-board micro-controller. The relative communication delay between the host computer and each robot is 100ms.

The control program platform allows for emulating communications between robots including communication topology. The testbed program platform is set up with three tiers. The top tier is the driver unit which tells the system the control references, communication topology, and the number of robots running in the system. The middle tier contains the controller for each robot which tells that robot how to actuate based on relative information gained from the neighbors. The third tier is composed of server units for each robot. These units have the primary responsibility for giving the second tier information upon request. These units also then take given control inputs and use them to actuate the physical robot.

The differential drive system of the ground robot can be described by the kinematics equations as

$$\dot{x}_i = v_i \cos(\psi_i), \tag{4.9}$$

$$\dot{y}_i = v_i \sin(\psi_i),$$

$$\dot{\psi}_i = \omega_i,$$

where x_i and y_i are the positions of the center of the robot, ψ_i is the orientation of rotation of the robot, and v_i and ω_i are the linear and angular velocities of the robot.

The robot has nonholonomic constraints shown as

$$\dot{x}_i \sin(\psi_i) - \dot{y}_i \cos(\psi_i) = 0, \tag{4.10}$$

which represents a simplified nonlinear constraint. To avoid using the nonlinear model of kinematics, the model is linearized by feedback linearization. A point that is L distance

away from the center is chosen at the front of the vehicle, this point is holonomic. In this manner, the holonomic point constructed as

$$x_{hi} = x_i + L_i \cos(\psi_i), \tag{4.11}$$

$$y_{hi} = y_i + L_i \sin(\psi_i),$$

where x_{hi} and y_{hi} represent the holonomic point L distance from the center of the vehicle. Therefore, the whole system can be described as

$$\begin{pmatrix} \dot{x_{hi}} \\ \dot{y_{hi}} \end{pmatrix} = \begin{pmatrix} \cos(\psi_i) & -L_i \sin(\psi_i) \\ \sin(\psi_i) & L_i \cos(\psi_i) \end{pmatrix} \begin{pmatrix} v_i \\ \omega_i \end{pmatrix}, \tag{4.12}$$

where the variables v_i and ω_i are control inputs.

4.5 Experiments and Results

4.5.1 Matlab Simulation (Continuous-Time)

Simulation results are separated into two controllers. The first controller, the single integrator Cartesian coupled controller, has control inputs of velocity. The second controller, the double integrator Cartesian coupled controller, has control inputs of acceleration. Both simulations are set up with four nonholonomic robots. The adjacency matrix given by

$$\begin{bmatrix} 0 & 0 & 1 & 0.5 \\ 1.2 & 0 & 0 & 0 \\ 0.1 & 0.9 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \tag{4.13}$$

will be used to describe the communication topology between vehicles.

i) 1st Order Coupled Coordinate Controller

For this controller simulation, three values of θ are chosen for the rotation matrix. The θ value is set to 0.7 radians which makes the motion convergent, and thus the robots spiral inward as in fig. 4.2(a). In all the following plots a circle represents the start of the simulation and a square denotes the end of the simulation.

The θ value is set to 1.2133 radians which makes the four robots follow each other in separate circular orbits, as in fig. 4.2(b).

The θ value is set to 1.35 radians, which makes the motion of the four robots spiral out as seen in fig. 4.2(c).

ii) 2nd Order Coupled Coordinate Controller

For this controller simulation, three values of θ are chosen for the rotation matrix. Additionally, each robot has an initial velocity during the start of the simulation. For the double integrator simulations and experiments, the initial velocity for $\dot{x}$ and $\dot{y}$ are shown as

$$\dot{x}_i(0) = 0.5, \tag{4.14}$$

$$\dot{y}_i(0) = 0.6,$$

where $\dot{x}_i(0)$ and $\dot{y}_i(0)$ represent the initial velocities for every agent within the system.

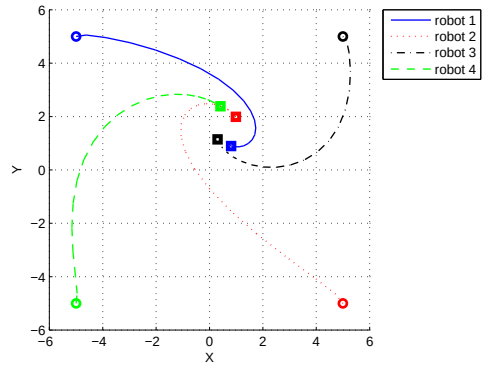
The θ value is set to 0.7 radians which makes the system convergent, and thus the robots spiral inward as in fig. 4.3(a). However, the motion of the robots converging will also travel in the direction of the initial velocities of the robots due to the controller design.

The θ value is set to 1.1 radians which makes the four robots follow each other in separate circular orbits along a motion given by the initial velocities, as in fig. 4.3(b).

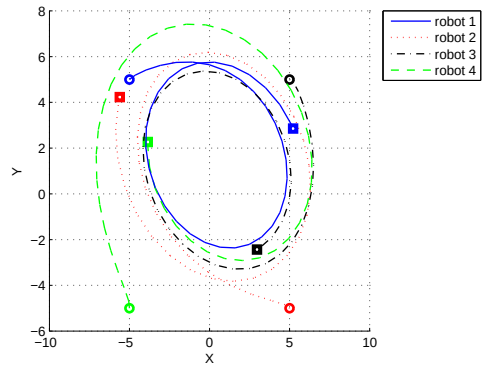
The θ value is set to 1.3 radians, which makes the system motion spiral out as in fig. 4.3(c).

4.5.2 AmigoBot Experiment (Discrete-Time)

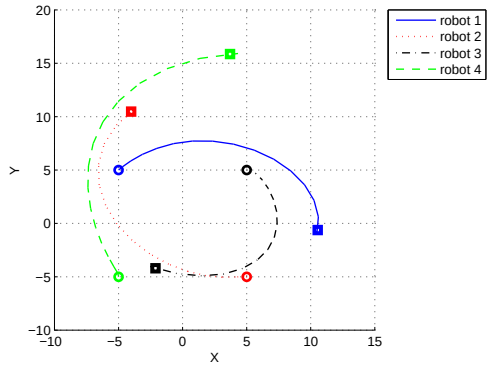
The above two controllers for the simulation were implemented in C code on the testbed program. The physical robots are very sensitive to gains in actuation, therefore a gain of



(a) 1st order controller with θ= 0.7 radians.

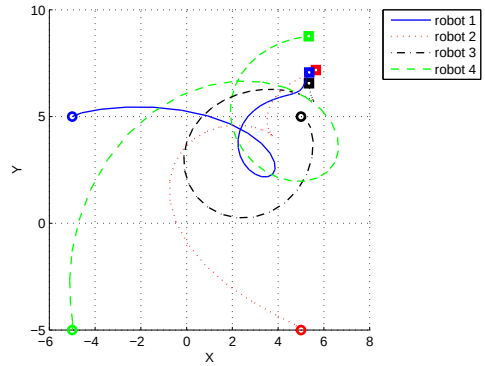


(b) 1st order controller with θ= 1.2133 radians.

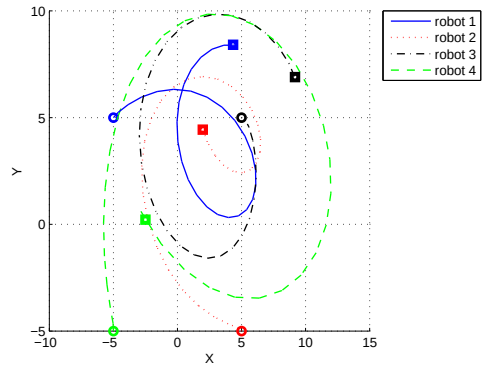


(c) 1st order controller with θ= 1.35 radians.

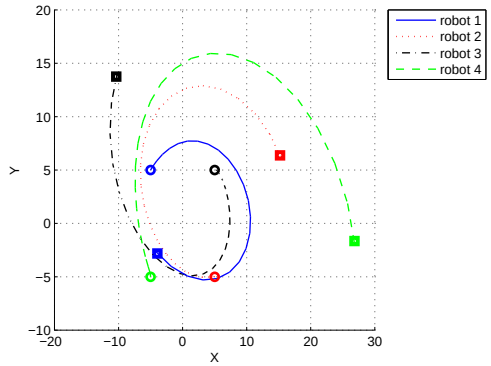
Fig. 4.2: Single integrator Matlab simulations.



(a) 2nd order controller with θ= 1.0 radians.



(b) 2nd order controller with θ= 1.1 radians.



(c) 2nd order controller with θ= 1.3 radians.

Fig. 4.3: Double integrator Matlab simulations.

0.001 was implemented to prevent control inputs from saturating the motors. The robots are also driven with both a velocity and angular velocity command. Therefore, on the double integrator Cartesian coupled controller, the values obtained for control inputs of acceleration are integrated before being sent to the robot. Although this results in not driving the robot directly from the control input, the controller calculates by retrieving the current velocity and position values and the results are integrated over a time period to develop the approximate velocity signal. For the continuous controllers shown above, the discrete form is very similar. However, there are some notable differences. The largest difference is that in the closed-loop control, instead of continuous feedback, the feedback obtained by a sample. As with all discrete controllers, the sampling time controls the stability of the system. Too large of a time results in an unstable system. Therefore within the Cartesian coordinate coupling controllers, both the sampling time and the Euler angle of the matrix play an integral role in controlling the stability of the system.

i) 1st Order Coupled Coordinate Controller

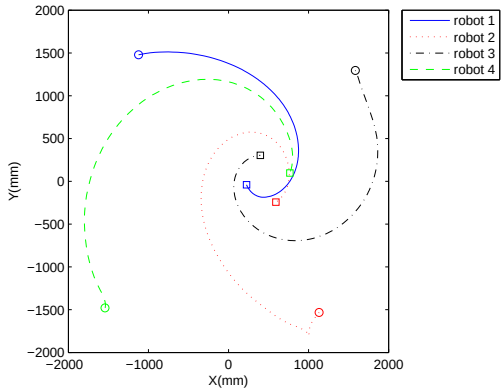
For this controller experiment, three values of θ are chosen for the rotation matrix. The θ value is set to 0.7 radians which makes the motion convergent and thus the robots spiral inward as in fig. 4.4(a). In the below plots a circle represents the start of the experiment and a square denotes the end of the experiment. Note: This data represents the actual encoder information from the robot, therefore upon convergence the data points do not meet because they represent the center of each physically spaced robot.

The θ value is set to 1.2133 radians which makes the four robots follow each other in separate circular orbits, as in fig. 4.4(b).

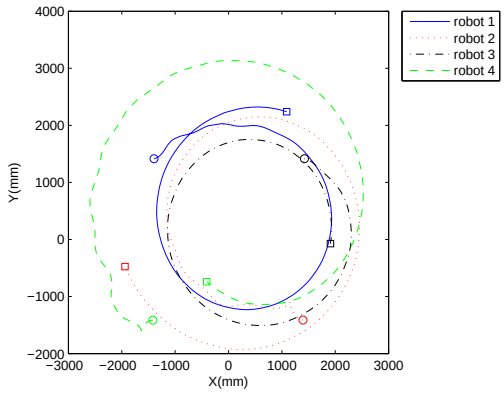
The θ value is set to 1.35 radians, which makes system motion spiral out as seen in fig. 4.4(c).

ii) 2nd Order Coupled Coordinate Controller

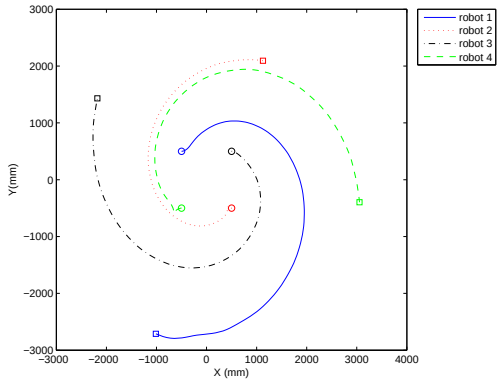
For this controller experiment, three values of θ are chosen for the rotation matrix. Additionally, each robot has an initial velocity as shown before in eq. (4.14).



(a) 1st order controller with θ= 0.7 radians.



(b) 1st order controller with θ= 1.2133 radians.



(c) 1st order controller with θ= 1.35 radians.

Fig. 4.4: Single integrator robot experimentation.

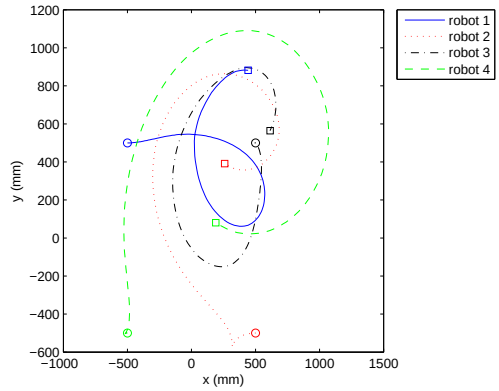
The θ value is set to 1.0 radians which makes the robots spiral inward as in fig. 4.5(a). However, the motion of the robots converging will also travel in the direction of the initial velocities of the robots due to the controller design.

The θ value is set to 1.1 radians which makes the four robots follow each other in separate circular orbits along a motion given by the initial velocities, as in fig. 4.5(b).

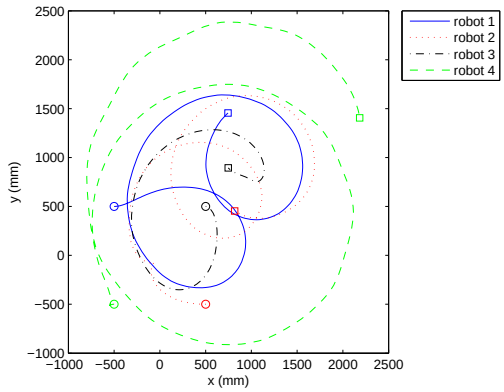
The θ value is set to 1.3 radians, which makes the system motion spiral out as in fig. 4.5(c).

4.6 Conclusion

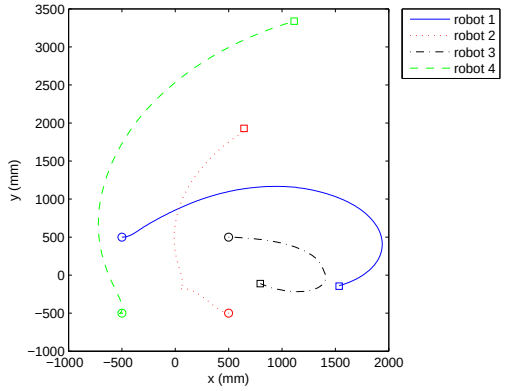
This section provides a study of the coordinated collective motion caused by using consensus based Cartesian coordinate coupling. The results provided by the simulation and actual experimentation prove that the controller works in a similar form as the simulations predict. The factors of communication topology, sampling period, system loop time delay, and the rotation matrix Euler angle all become prevalent in controlling the stability of the system, and thus the collective motion of the system. This study was proven not only with single integrator kinematics, but it was also shown to work similarly in the double integrator dynamic relative damping controller. The application of one single value of an Euler angle provides extreme control in creating logarithmic spiral convergence and divergence, along with marginally stable orbits. This single angle entry provides for a group of decentralized robots to be controlled in complex collective motions without the need for hugely complex code or increased computation.



(a) 2nd order controller with θ= 1.0 radians.



(b) 2nd order controller with θ= 1.1 radians.



(c) 2nd order controller with θ= 1.3 radians.

Fig. 4.5: Double integrator robot experimentation.

Chapter 5

Future Combination and Contribution

5.1 Introduction and Motivation

There are several problems which require the application point or viewing perspective of an actuator to be modified to best effectively find a solution. No more is this true then when fighting a fire or chemical spill. The spray of water or agent must be spread over a large area of the problem in an effective manner. To avoid wasting valuable supplies, that agent must be used to cover all of the area required. The application of water or some other product must also be adjustable and adaptable to the situation as it changes.

In a similar manner, when a group of UAVs are patrolling an area their area of coverage and overlapping coverage capability become extremely important so as not to miss an important event or issue. For a ground vehicle, this would apply in importance to vision mapping. Stereo vision is used to give the user a view of an environment in a quasi-3d perspective. This is done by using two cameras that are spaced apart similar to the human eyes. As the perspective of these images will be different, they will provide a manner of dimensional information. When these images are combined later, the user can view the environment filmed as if it were almost 3-dimensional. However, a downfall exists in that the images are only 3-dimensional by two perspectives. Therefore, to boost the dimensionality of the image, the taking of an environment with a stereo camera can greatly be improved if those images are then taken from multiple different perspectives together. With many perspectives all taken through stereo imaging, computer software can then combine those images to provide more detail of the terrain. Although this is not as powerful as ladar would be, it provides more information than a static perspective of a stereo camera might.

In Chapters 2 and 3, a robot arm and robot arm network was developed that could be used to coordinate robot arms through consensus. When demonstrating the robot arm

network previously, the robot arms were coordinated to synchronize to the same angles of actuation. However, those same robot arms can also be synchronized to follow a time varying reference. Additionally, each arm can be set to remain at an offset of its neighbors. By controlling the amount of offset and the varying signal, the system can be designed to allow the arms to oppose each other or synchronize in a multitude of motions. When spraying down an area through robot arm actuation, such a method of opposing arm consensus would be useful in applying varying chemical substances one after another. In fire fighting, it is common that one line of water spray fans the fire out while another stream is used to diminish the fanning fire. By having consensus throughout the group, but allowing for adjustable offsets between arms, the arms could conceivably be designed to fight fires as efficiently as humans do. That same opposing offset nature could also be applied to using a camera on the end effector. As one camera sweeps one field area, an opposing arm sweeps the uncovered area left from the previous. This motion would continue in offsets until all areas are covered. In this section, this idea will be explored to understand how consensus controlled robot arms could be used on ground robots.

In Chapter 4, the study of Cartesian coordinate collective motion was examined and validated. It was presented that Cartesian coordinate collective motion can drive the cyclic motion of the vehicle system. By varying the Euler angle from small to large, the system will perform the behaviors of spiralling in and out, or orbiting. This motion allows for the coverage of a ground area as the system can be adjusted to spiral in or out. While this alone applies to ground coverage and possible applications to the goals stated above, the final application below will combine previously stated methods into a simulation that explores how that coverage could further be expanded.

5.2 Simulated Example

The following represents an example of a ground vehicle and arm system that could be used to either film its environment for mapping or patrolling surveillance. However, the following example could also be applied to chemical actuation of a dangerous threat such as a gas or fire. The simulation will combine the two parts stated in the introduction.

Using Matlab, a system of ground robots will me modeled that are similar to that of the actual AmigoBot system. Additionally, robot arms will be attached to these vehicles with similarities to that of physical robot arms.

5.2.1 Robot Arm Kinematics

For this robotic arm simulation, it was desired to have a nonlinear coupled two-joint system. A two-joint system is chosen to represent the kinematics of a robot arm while maintaining simplification in study. There are several values that need to be addressed. The length of the robot arm is separated into l_1, which represents the first joint of the arm, and l_2, which represents the second joint. Each arm has inertia of $I1$ and $I2$ and a gravitational force of $m_1 g$ and $m_2 g$. The angle of the first arm is in respect to the ground plane and is represented as q_1. The angle of the second arm is based off of the plane tangent created from the first arm and is represented as q_2.

The robotic arm system is described as a relationship of the angular arm acceleration, velocities, and arm torque. The system equation [22] can be described as

$$M(q)\ddot{q} + C\dot{q} + G = \tau, \tag{5.1}$$

where the values of $\ddot{q}$, $\dot{q}$, and τ describe angular acceleration, velocity, and torque. For the system model, the input into the arm is torque the output is angular acceleration and velocity. Each $q = [q_1, q_2]^T$ represents the angle combination. The value of $M(q)$ as

$$M(q) = m_1 J_{vc1}^T J_{vc1} + m_2 J_{vc2}^T J_{vc2} + \begin{bmatrix} I1 + I2 & I2 \\ I2 & I2 \end{bmatrix}, \tag{5.2}$$

which describes the inertial matrix. For simplification purposes, this matrix is expanded as

$$d_{11} = m_1 l_{c1}^2 + m_2(l_1^2 + l2_{c2} + 2l_1 l_{c2}\cos(q_2) + I1 + I2, \tag{5.3}$$

$$d_{12} = d_{21} = m_2(l_{c2}^2 + l_1 l_{c2}\cos(q_2) + I2,$$

$$d_{22} = m_2 l_{c2}^2 + I2,$$

where each equation represents one of the four index values of the original matrix. Inside the system description (5.2), the matrix C can be found as

$$C = \begin{bmatrix} h\dot{q}_2 & h\dot{q}_2 + h\dot{q}_1 \\ -h\dot{q}_1 & 0 \end{bmatrix}, \tag{5.4}$$

which acts upon $\dot{q}$, where $h = -m_2 l_1 l_{c2} \sin(q_2)$. Finally from system (5.2), is the description of gravity or G can be found as

$$g_1 = (m_1 l_{c1} + m_2 l_1) g cos(q_1) + m_2 l_{c2} g cos(q_1 + q_2), \tag{5.5}$$

$$g_2 = m_2 l_{c2} g cos(q_1 + q_2),$$

where g_1 and g_2 are stacked in a column matrix to represent G.

5.2.2 Single Order Cartesian Coordinate Coupling Ground Control

The control of the robotic system is based on comparing the position data and velocity data of robot i with the data of the robots neighbors using consensus. The robots neighbors are determined by the adjacency matrix that determines whether the robot can receive data from its neighbor or not. This adjacency matrix A_{ij}, can be a simple 1 or 0 index valued matrix or more determined by individual gains that the ith robot can communicate with to the jth neighbor as explained previously in this thesis. The control input is based on a summation of the differences for position and position with a reference. Additionally, within the summation for position is a rotation matrix. Such a 2-dimensional rotation matrix can be found as

$$R_{\theta_{cc}} = \begin{bmatrix} cos(\theta_{cc}) & -sin(\theta_{cc}) \\ sin(\theta_{cc}) & cos(\theta_{cc}) \end{bmatrix}, \tag{5.6}$$

which creates the previously described Cartesian coordinate controller. The addition of the rotation matrix develops the stability control of this controller. At very small values of θ_{cc}, the robots will behave by spiraling into a convergence of positions. If the value of θ_{cc} is too large, the system will be forced more unstable and the robots will spiral outward. At some value of θ_{cc}, the rotation matrix will force the eigenvalues of the system to behave at a marginal stability. This causes the robots to rotate in a circle with each other as they are on the edge of spiral in or out. The total control input into the differential drive robot is found as U_{x1} and U_{x2} using

$$U_i = -\sum_{j=1}^{n} A_{ij} R_\theta P_1(\psi_i - \psi_j) + P_2(\psi_i - Ref), \tag{5.7}$$

where gains of P_1 and P_2 are used to change the hypotrochoid behavior of the system, the value of ψ is the state of the system, and i is the acting robot and the state representing the Cartesian position of the robot. These values can be considered to be the linear and angular velocity for the system. The reference value for this example adds a final level of complexity to the controller in the form of a circle. If the Cartesian coordinate controller is set to allow the marginally stable cyclic orbits, then the combination along with a circular reference will create what is known as a hypotrochoid. A hypotrochoid allows for complex ground coverage that shifts at each revolution of the system, which will present interest in the goals for ground coverage.

5.2.3 Robot Arm Control

The control input into the robotic arm is the torque of the two arm joints. This control input is based on the angular positions of the ith robot with its jth neighbor at the same joint. Additionally, the ith robot is also compared with a reference. The controller also has a summation to take into account consensus of the angular velocities of $\dot{q}$, which will better

prepare the controller to handle a time varying reference. To provide for accurate physics the inertia of the second arm with the angular velocities of the arm takes in account for the coupling behavior. The robotic arm system control can be described by

$$U_i = -\sum_{j=1}^{n} F_{ij}(q_i - q_j) + (q_i - Ref_{2i}) - \sum_{j=1}^{n} F_{ij}(\dot{q}_i - \dot{q}_j) - I2\dot{q}_i, \tag{5.8}$$

where the value of F_{ij} is the adjacency matrix representative of the robot arm network testbed and Ref is the time varying signal. In this robot arm reference setup, the second joint of the robot arm does the main sweeping of actuation. The first joint, which is mechanically coupled to the second, serves to sweep the elevation back and forth of the main second arm sweep. The result of these two sweeping motions, allows for flexibility in the area of coverage in height, span, and resolution that the robot arm can make. The reference for this example is based on whether the arm is even or odd in address value. If the robotic arm is even the reference is a cosine function based on time. Conversely, the function is a sine function if the number is odd. This behavior creates the sweeping nature of the robotic arms. To ensure that the coverage of the system is more uniform, half the arms in an even system will be cosine and half sine. Control of the reference's division of time can control the compactness of the sweep whether for actuation or vision. In this design, assuming that the robot arms will be in motion, this creates the ability to control the region of a room in which the robot arm end effector covers through use of gains and reference values.

5.2.4 Results

For this simulation, four ground robots with arms were used in Matlab and Simulink. The θ_{cc} for the Cartesian coupling rotation matrix is set to 1.57 radians which makes this particular system, and thus the robots will rotate in a circle clockwise. Additionally, the group of ground robots follow a circle equation as a reference. By varying the Euler angle for the rotation matrix, a hypotrochoid of varying radius and spacing can also be created. Finally, the robotic arms are set up as described above and the arms are split between

cosine and sine references.

For this simulation, the robotic arms are set up to point perpendicular to the travel of the vehicle and thus outwards from the circular motion. To better understand the various motions involved, the motions of the sweeping arms are plotted in fig. 5.1. Notice that half follow a cosine function and half a sine function. The velocities of the sweeping arms can also be seen in fig. 5.2. Note that the velocities do not follow as well as the positions due to the physical kinematics of the robot arm.

To understand the kinematic coupling effect of the sweeping arms on the base elevation arms, the plots for position and velocity of the first joint can be seen in figs. 5.3 and 5.4.

It can be seen that the sinusoidal behavior of the sweeping second joint inertia has an impact on the first jointed arm. Note that the controller tries to dampen the behavior to remain in consensus.

Finally, fig. 5.5 describes the combined motion of position through time of the ground robot motion. This motion is very similar to the hypotrochoid mathematical equation. This is a combination of the robots spinning around a central point and that central point following a circle reference.

5.3 Conclusions

The hypotrochoid behavior of the robot bases combined with the sweeping behavior of the robot arms provides a wonderful ability to change perspective or actuation within an environment. By using consensus, the effort caused by each robot can be shared and minimized. Consensus can simplify the creation of complex motions of the robotic system. The current simulation allows for change in sweeping angle, elevation, and coverage with the motion of the ground robots. By using consensus, the robots do not all have to have the reference, and thus the limits of delay within the system are also minimized. Finally, the additional ability to tune the stability of the inner circular motion allows for distinct possibilities in changing the hypotrochoid to a different area of coverage. Thus, by combining the studies of this thesis for both a robot arm network and collective motion, it is shown that there are a range of applications that can use consensus for actuation and for

general motion. Such an example has many gains, values, and angles that can be changed to vary the coverage or behavior of the system as a whole. While this chapter purely focuses on a simulation that integrates the implementations already performed within this thesis, future goals exist to implement this simulation in reality as a further tool to expand the motion control and ability of a robot. Further expansion will lead to a larger measure of capabilities that any tasked robot system can draw upon to solve problems with.

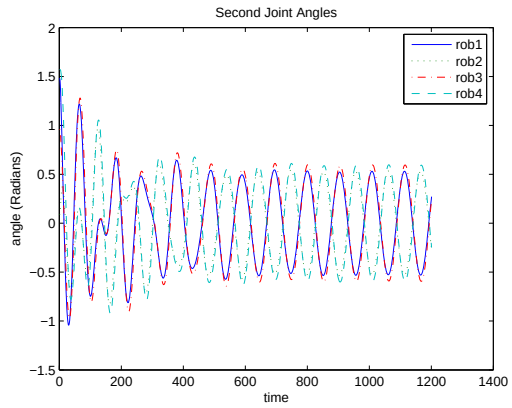


Fig. 5.1: Sweeping joint angles.

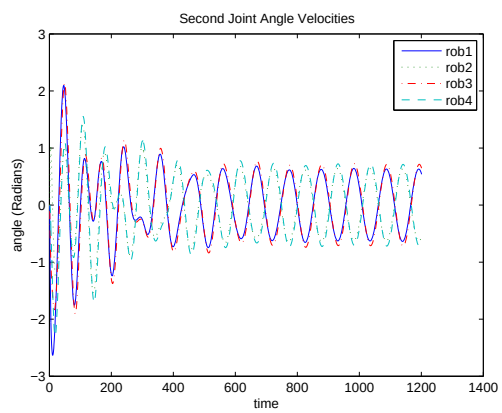


Fig. 5.2: Sweeping joint velocities.

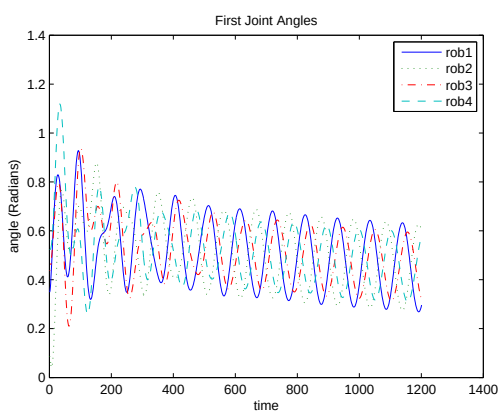


Fig. 5.3: First joint angles.

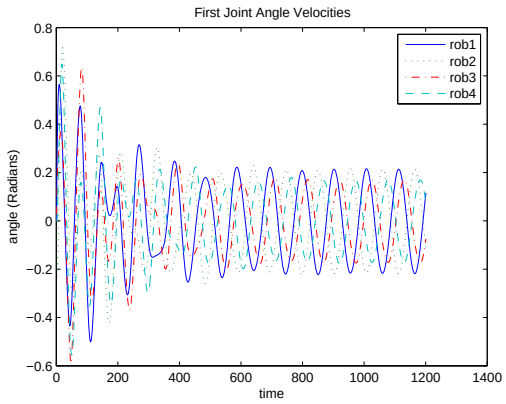


Fig. 5.4: First joint velocities.

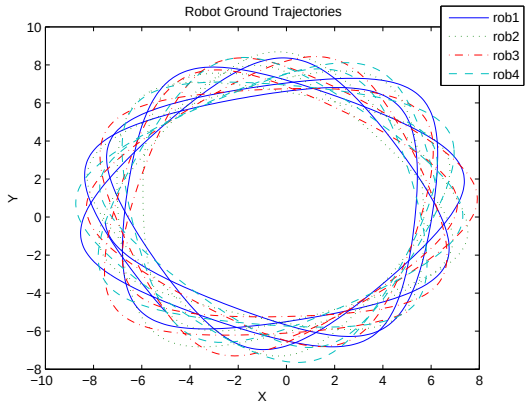


Fig. 5.5: Robot ground trajectories.

Chapter 6

Conclusion

6.1 Summary of Results

The purpose of the research presented within this thesis is to implement and validate several control algorithms and platforms in the area of consensus based multi robot group control. Through this thesis, several platforms for research have been explored and created. Several new and interesting controllers have also been studied and presented to work in the physical world.

In preparation and creation of a more powerful ground robot and for use in a network testing platform, two robot arms were created and presented in Chapter 2. The first arm explored the many possibilities and capabilities using a open source project developed within the hobbyist community. Although this arm proved to work well, several issues such as build time and unproven circuitry drove the research to find an alternative solution. As such, the decisions and conclusions of choosing a better arm were discussed within this chapter. Finally, this chapter presented a second robot arm that could be built in large quantities efficiently. A platform of software for both arms was created, and functions were designed for both implementation with a robotic arm network and for future research goals that may arrive later within the lab research.

In Chapter 3, the combination of a robot arm platform created within Chapter 2 is expanded upon to implement a test bed for consensus based robotic arm research. As much has been done with mobile robot vehicles within consensus, adding consensus to mobile or static robot arms will improve capabilities a robot has in solving problems. Within this chapter, the platform of both software and hardware is presented and the results prove that the platform can run consensus algorithms and log the results of that data. This platform will prove useful in future research within the lab, whether through consensus or some other

control theory combination.

In Chapter 4, two types of collective motion controllers are discussed. Both controllers are coupled versions of the basic consensus equation. The coupling of Cartesian coordinates is done through the addition of a rotation matrix to a standard consensus equation. The addition of the rotation matrix offers one more variable that controls system stability and also controls collective motion of the robot group. The Euler angle of the rotation matrix rotates the eigenvalues of the system's Laplacian effecting stability. Based on the Euler angle, the system will spiral towards convergence in a stable manner, cyclically pursue in various orbits through a marginal stability, or slowly spiral outward in an unstable rate. Within this chapter, the AmigoBot platform at its current nature is presented and both controllers are demonstrated through simulation for reference. One controller offers control of a single integrator kinematic system, while the other controller focuses on relative damping of a double integrator dynamic system. Both controllers are presented and implemented on the AmigoBot platform testbed, and results are offered in comparison to continues time simulation results.

In Chapter 5, the two separate research topics from this thesis are combined. Utilizing the knowledge gained through the implementation of a robotic arm network testbed from Chapter 3, a simulation of sweeping robot arms was presented. These robot arms sit on top of ground robots which are free to travel in their own motion. Utilizing the knowledge gained from Chapter 4 on Cartesian coupled collective motion, the ground robots are simulated with a single order controller which is simulated with a reference. By putting in the reference of a circle through time, the system of ground robots follow two circular paths culminating in the motion of a mathematic hypotrochoid. This chapter provides several examples of applications that can use both the robotic arm network and the collective motion together. A simulated example is offered as one of many examples of how this could take place.

6.2 Future Work

Within the robot arm implementation presented in Chapter 2, there is a great deal of room to explore adding functionality to prevent the robot arm from fighting issues with

overloading motors and heavy load pickup. There is also a great deal of room to explore consensus topics and theories using both velocity feedback and torque, which can both be calculated or estimated from each robot arm joint. Much of this research would also add to the capabilities of the robotic arm network presented in Chapter 3.

The robot arm network in Chapter 3 suffers from a great deal of lack of optimization. This is partially due to the lack of time to explore better methods of functionality. However, future work on this network could greatly optimize time delays between each robot arm within the network and in serial communication with the robot arm and its various modules. Additionally, it would be desired to add stronger organization to how the threads access and move data around within the program. Currently, the threads operate on almost a free for all manner. Adding some strict organization and priority to how data is accessed over the network and within each program would most likely speed up the overall capability of the network and ensure that a larger number of robot arms could be handled under the platform.

The two controllers studied within Chapter 4 offered a good deal of insight to the capabilities of consensus with collective motion. Although the controllers functioned well in the discrete time system, the addition and improvement of how the controllers are implemented may make the results better. Currently, the double integrator control output is integrated to get velocity commands that could be used and interpreted within the AmigoBot. This ultimately adds some estimation error to the final output. By redesigning current AmigoBot platform, the ability to control accelerations and torque would probably offer better results, however the results currently are reasonable and prove that the theory is true.

Finally, all of Chapter 5 represents work that can be put into implementation in the future. Of particular interest will be to actual integrate the collective motion controllers along with the robotic arm network testbed. The goal will be to validate the simulation results presented within this thesis. Additionally, the motions within this thesis, along with the many other motions previously learned within the lab will be combined to offer a robot that can provide search and rescue type capabilities, while utilizing collective motion and

robot arm techniques to best perform individual goals and tasks that it would typically face within such a challenge.

6.3 Conclusions

Three different testbed platforms have been developed within this thesis. The first two are different robot arm platforms that provide varying functionality and capability. These will both be useful in future research with the lab. The robot arm platforms provide our ground robots with increased operation ability and span. Additionally, the robot arm platform was combined into a robot arm network testbed platform. This platform allows for a great deal of future research within both consensus and possible nonconsensus based control theory within a group of robot arm. This implementation also improves the complete capability of our current mobile system and provides for potential expanded research and applications. Two different collective motion controllers have been experimentally validated. Both controllers prove to work in similarity to simulation. The controllers will provide to the overall capability of the lab and the future application and capability of both ground and aerial robot systems. Finally, several applications and a simulated example have been provided to spark interest in the capabilities that the research within this thesis can add to group robotics to fix problems that need to be solved.

References

[1] P. Wang, "Navigation strategies for multiple autonomous mobile robots moving in formation," *IEEE/RSJ International Workshop on Intelligent Robots and Systems*, pp. 486–493, 1989.

[2] F. Arrichiello, *Coordination Control of Multiple Mobile Robots*. Ph.D. dissertation, Universita Degli Studi Di Cassino, Italy, 2006.

[3] K.-H. Tan and M. A. Lewis, "High-precision formation control of mobile robots using virtual stuctures," *Autonomous Robots*, vol. 4, pp. 387–403, 1997.

[4] T. Balch, R. C. Arkin, and S. Member, "Behavior-based formation control for multi-robot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1999.

[5] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, pp. 215–233, 2007.

[6] J. Lin, A. Morse, and B. Anderson, "The multi-agent rendezvous problem," *2003 Proceedings. 42nd IEEE Conference on Decision and Control*, vol. 2, pp. 1508–1513, 2003.

[7] W. Ren and R. W. Beard, *Distributed Consensus in Multi-vehicle Cooperative Control*, ser. Communications and Control Engineering. London: Springer-Verlag, 2008.

[8] "Mobile robots inc," Available: http://www.mobilerobots.com.

[9] S. Ziauddin and A. Zalzala, "Cooperative control of multiple arms mounted on a mobile platform," *1996 UKACC International Conference on Control*, vol. 1, no. 427, pp. 341–346, 1996.

[10] J. Tan and N. Xi, "Integrated sensing and control of mobile manipulators," *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 865–870, Nov. 2001.

[11] W. Ren, "Collective motion from consensus with cartesian coordinate coupling," *IEEE Transactions on Automatic Control*, vol. 54, no. 6, pp. 1330–1335, June 2009.

[12] W. Ren, H. Chao, W. Bourgeous, N. Sorensen, and Y. Chen, "Experimental validation of consensus algorithms for multi-vehicle cooperative control," *IEEE Transactions on Control Systems Technology*, vol. 16, no. 4, pp. 745–752, July 2008.

[13] W. Ren, "Collective motion from consensus with cartesian coordinate coupling - part i: Single-integrator kinematics," ser. IEEE Conference on Decision and Control, pp. 1006–1011, Cancun, Mexico, Dec. 2008.

[14] W. Ren, "Collective motion from consensus with cartesian coordinate coupling - part ii: Double-integrator dynamics," ser. IEEE Conference on Decision and Control, pp. 1012–1017, Cancun, Mexico, Dec. 2008.

[15] "Open servo project- open source hardware and software," Available: http://www.OpenServo.org.

[16] "Lynxmotion robotics," Available: http://www.lynxmotion.com.

[17] "Lego inc," Available: http://www.lego.com.

[18] B. Carter, "Robotfuzz," Available: http://www.robotfuzz.com/.

[19] "Avrdude bootloader and compiler," Available: http://www.bsdhome.com/avrdude/.

[20] "Robotis robots," Available: http://www.robotis.com.

[21] P. Teodoro, "Humanoid robot: Development of a simulation environment of an entertainmnet humanoid robot," Master's thesis, Universidade Tecnica de Lisboa, Sept. 2007.

[22] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. New York: John Wiley and Sons, Inc, 2006.

[23] T. Schneider, "Tserial communication class," Available: http://www.tetraedre.com/advanced/serial/.

[24] A. Prior, "Practical sockets communication class," Available: http://www.csse.uwa.edu.au/ tone/.

Appendices

Appendix A

Open Servo Control Registers

address	name
0x00	DEVICE_TYPE
0x01	DEVICE_SUBTYPE
0x02	VERSION_MAJOR
0x03	VERSION_MINOR
0x04	FLAGS_HI
0x05	FLAGS_LO
0x06	TIMER_HI
0x07	TIMER_LO
0x08	POSITION_HI
0x09	POSITION_LO
0x0A	VELOCITY_HI
0x0B	VELOCITY_LO
0x0C	POWER_HI
0x0D	POWER_LO
0x0E	PWM_CW
0x0F	PWM_CCW
0x10	SEEK_HI
0x11	SEEK_LO
0x12	SEEK_VELOCITY_HI
0x13	SEEK_VELOCITY_LO

0x14	VOLTAGE_HI
0x15	VOLTAGE_LO
0x16	CURVE_RESERVED
0x17	CURVE_BUFFER
0x18	CURVE_DELTA_HI
0x19	CURVE_DELTA_LO
0x1A	CURVE_POSITION_HI
0x1B	CURVE_POSITION_LO
0x1C	CURVE_IN_VELOCITY_HI
0x1D	CURVE_IN_VELOCITY_LO
0x1E	CURVE_OUT_VELOCITY_HI
0x1F	CURVE_OUT_VELOCITY_LO
0x20	TWI_ADDRESS
0x21	PID_DEADBAND
0x22	PID_PGAIN_HI
0x23	PID_PGAIN_LO
0x24	PID_DGAIN_HI
0x25	PID_DGAIN_LO
0x26	PID_IGAIN_HI
0x27	PID_IGAIN_LO
0x28	PWM_FREQ_DIVIDER_HI
0x29	PWM_FREQ_DIVIDER_LO
0x2A	MIN_SEEK_HI
0x2B	MIN_SEEK_LO

0x2C	MAX_SEEK_HI
0x2D	MAX_SEEK_LO
0x2E	REVERSE_SEEK
0x2F	RESERVED
0x00...0x7F	reserved
0x80	RESET
0x81	CHECKED_TXN
0x82	PWM_ENABLE
0x83	PWM_DISABLE
0x84	WRITE_ENABLE
0x85	WRITE_DISABLE
0x86	REGISTERS_SAVE
0x87	REGISTERS_RESTORE
0x88	REGISTERS_DEFAULT
0x89 0x90 0x91	CURVE_MOTION_ENABLE
0x92	CURVE_MOTION_DISABLE
0x93	CURVE_MOTION_RESET
0x94	CURVE_MOTION_APPEND

Appendix B

Bioloid Arm Control Registers

address	name
0x00	Model Number(L)
0x01	Model Number(H)
0x02	Version of Firmware
0x03	ID
0x04	Baud Rate
0x05	Return Delay Time
0x06	CW Angle Limit(L)
0x07	CW Angle Limit(H)
0x08	CCW Angle Limit(L)
0x09	CCW Angle Limit(H)
0x0A	(Reserved)
0x0B	Highest Limit Temperature
0x0C	Lowest Limit Voltage
0x0D	Highest Limit Voltage
0x0E	Max Torque(L)
0x0F	Max Torque(H)
0x10	Status Return Level
0x11	Alarm LED

0x12	Alarm RD
0x13	(Reserved)
0x14	Down Calibration(L)
0x15	Down Calibration(H)
0x1E	Goal Position(L)
0X1F	Goal Position(H)
0x20	Moving Speed(L)
0x21	Moving Speed(H)
0x22	Torque Limit(L)
0x23	Torque Limit(H)
0X24	Present Position(L)
0X25	Present Position(H)
0X26	Present Speed(L)
0X27	Present Speed(H)
0X28	Present Load(L)
0X29	Present Load(H)
0X2A	Present Voltage
0X2B	Present Temperature
0x2C	Registered Instruction

Appendix C

Robot Arm Comparison Chart

The following chart is a representation of the various off the shelf robot arms explored. Many of these robot arms represent a whole series of options that all fall within the same specifications and costs. This is provided to show the spectrum of robot arms available, along with how each arm meets the required constraints, and why the decision was made to design an arm instead of purchasing one.

Name	Company	D.O.F	Weight	Control	Feedback	Price
AL5A	Lynxmotion	4	23 oz	Serial	None	$ 338.65
P-series	Mobile Robots	7	50 oz	Serial	None	$ 6000
HS-series	Denso	4	2000 oz	Serial	Position, Velocity	$ 3500
Pioneer Arm	Mobile Robots	5	50 oz	Serial	None	$ 6000
Scorbot-ER9	Intelitek	5	2000 oz	USB	Position	$ 30,000
Scorbot-ER4	Intelitek	5	400 oz	USB	Position	$ 8000
Robotic Arm Edge	OWI	4	23 oz	USB	None	$ 150