ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Motion Planning and Control of an Omni-directional Mobile Robot with Caster Wheels

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΛΑΝΤΟΣ

Επιβλέπων : Κωνσταντίνος Σ. Τζαφέστας
Επίκ. Καθηγητής Ε.Μ.Π.

Faïz Ben Amar
Maître de Conférences U.P.M.C.

Αθήνα, Ιούλιος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Motion Planning and Control of an Omni-directional Mobile Robot with Caster Wheels

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΛΑΝΤΟΣ

Επιβλέπων : Κωνσταντίνος Σ. Τζαφέστας
Επίκ. Καθηγητής Ε.Μ.Π.

Faïz Ben Amar
Maître de Conférences U.P.M.C.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Ιουλίου 2011.

....	....	....
Κωνσταντίνος Σ. Τζαφέστας	Κιαμάλ Πεκμεστζή	Κυριακόπουλος Κωνσταντίνος
Επίκουρος Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2011

....
Κωνσταντίνος Λάντος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η παρούσα διπλωματική εργασία ασχολείται με θέματα που αφορούν τα κινητά (mobile) ρομπότ όπως ο εντοπισμός θέσης (localization) του κινούμενου ρομπότ και η πλοήγηση του (navigation) στο χώρο, και ειδικότερα ο σχεδιασμός τροχιάς και ο έλεγχος της κίνησης του ρομπότ. Ειδικότερα μελετώνται τα κινητά ρομπότ που ονομάζονται πανκατευθυντικά (omni-directional). Ακόμη παρουσιάζονται θέματα προγραμματισμού μικρο-ελεγκτών, ασύρματης επικοινωνίας μεταξύ τους, επικοινωνίας με υπολογιστή και ανάπτυξης προγράμματος διεπαφής χρήστη.

Στην αρχή της εργασίας γίνεται μια εισαγωγή στα omni-directional ρομπότ και μια αναφορά στην βιβλιογραφία και στα ήδη υπάρχοντα συστήματα. Αναλύονται οι έννοιες που θα μας απασχολήσουν, τα προβλήματα που προέκυψαν στην διαδικασία και η μεθοδολογία που ακολουθήσαμε. Τα omni-directional ρομπότ είναι συστήματα που έχουν την δυνατότητα να κινούνται προς κάθε κατεύθυνση χωρίς να χρειάζεται να αλλάξουν τον προσανατολισμό τους. Αυτή η δυνατότητα δίνεται στο σύστημα από ειδικού τύπου τροχούς, όπως αναλύεται στην εργασία. Στην παρούσα εργασία επιλέχθηκαν τα omni-directional ρομπότ με caster-type τροχούς. Αυτού του είδους οι τροχοί παρουσιάζουν αρκετά πλεονεκτήματα σε σχέση με τους υπόλοιπους και είναι πολύ πιο εύχρηστοι.

Σκοπός της εργασίας είναι η ανάπτυξη ενός συστήματος ελέγχου omni-directional ρομπότ με διεπαφή χρήστη. Το σύστημα πρέπει να περιλαμβάνει αλγορίθμους εντοπισμού θέσης (localization) και σχεδιασμού τροχιάς (path generator), οδήγηση κινητήρων και επικοινωνία με αισθητήρες. Ακόμη, το σύστημα πρέπει να είναι ευέλικτο ώστε να μπορεί να εφαρμόζεται σε όλων των ειδών τις συνθέσεις omni-directional ρομπότ με caster τροχούς (δύο τροχών, τριών τροχών κτλ) χωρίς ιδιαίτερες αλλαγές στον προγραμματισμό. Ο σχεδιασμός του συστήματος πρέπει να γίνει σε ανεξάρτητες μονάδες οι οποίες θα επικοινωνούν μεταξύ τους. Έτσι, θα μπορούμε να προσθέτουμε μεθόδους localization ή path generator, με συνέπεια ως προς το πρωτόκολλο επικοινωνίας με τα υπόλοιπα κομμάτια του συστήματος, χωρίς να χρειάζεται να αλλάξουμε κάτι άλλο στο σύστημα.

Αρχικά το σύστημα σχεδιάστηκε για ένα omni-directional ρομπότ με τρεις caster τροχούς. Στην εργασία αναπτύσσεται το κινηματικό μοντέλο αυτού του ρομπότ, που όμως μπορεί εύκολα να τροποποιηθεί ώστε να εφαρμοστεί σε άλλου είδους συνθέσεις omni-directional ρομπότ με caster τροχούς. Το κινηματικό μοντέλο θεωρεί ότι κάθε τροχός είναι ανεξάρτητος από τους υπόλοιπους και εκφράζει τις εξισώσεις κίνησης ως προς το κέντρο του ρομπότ. Αυτό δίνει την δυνατότητα να έχουμε όσους τροχούς θέλουμε και σε όποια θέση. Με βάση αυτό το κινηματικό μοντέλο αναπτύσσουμε μεθοδολογίες υπολογισμού οδομετρίας για τον εντοπισμό θέσης (localization) του ρομπότ.

Το σύστημα αποτελείται από διάφορες συνιστώσες. Τα κυριότερα από αυτά είναι το πρόγραμμα των μικροελεγκτών PIC που οδηγούν τους κινητήρες στους τροχούς και το πρόγραμμα διεπαφής χρήστη που εκτελείται σε υπολογιστή με περιβάλλον Windows που βρίσκεται πάνω στο ρομπότ. Ακόμα και τα ίδια τα προγράμματα αποτελούνται από λογισμικές μονάδες, όπως ασύρματη και ενσύρματη επικοινωνία, επικοινωνία με αισθητήρες, υπολογισμός οδομετρίας κ.ά. Αυτός ο διαμερισμός του συστήματος το καθιστά περισσότερο ευέλικτο. Αλλάζοντας μια συνιστώσα και κρατώντας σταθερά τα πρότυπα εισόδου/εξόδου, η αλλαγή είναι απόλυτα διαφανής στο ολικό σύστημα.

Παράλληλα, αναπτύχθηκε μια προσομοίωση του ρομπότ σε περιβάλλον Matlab για την μελέτη διαφόρων αλγορίθμων σχεδιασμού τροχιάς (path generators). Ζητούμενο είναι η επιλογή ενός αλγόριθμου που θα χρησιμοποιηθεί στο σύστημα, ο οποίος θα υπολογίζει το ζητούμενο προφίλ κίνησης (θέσης-ταχύτητας) που θα δίνεται ως είσοδος στο σύστημα. Κριτήρια για την επιλογή του αλγορίθμου είναι η δυνατότητα επιλογής μέγιστης ταχύτητας αλλά και κίνηση στην μεγίστη ταχύτητα για το μεγαλύτερο μέρος της κίνησης. Τέλος, προσομοιώνουμε το ρομπότ και βλέπουμε πώς το κινηματικό μοντέλο συμπεριφέρεται σε κάθε αλγόριθμο.

Το σύστημα εφαρμόζεται τελικώς σε δύο ρομπότ, ένα με τρεις τροχούς και ένα με δύο τροχούς (στην πραγματικότητα είναι με τέσσερις τροχούς, αλλά μόνο οι δύο είναι ενεργοί, οι άλλοι είναι μόνο για θέματα σταθερότητας) ώστε να ελεγχθεί η ευελιξία του. Μελλοντική εργασία αφορά στην πραγματοποίηση πειραμάτων για μέτρηση και επικύρωση της οδομετρίας ή και αναζήτηση καλύτερων μεθόδων εντοπισμού θέσης (localization) και σχεδιασμού τροχιάς (path planning) που μπορούν να εφαρμοστούν στο σύστημα.

Λέξεις Κλειδιά

Πανκατευθυντικό ρομπότ, έλεγχος ρομπότ, σχεδιασμός τροχιάς, εντοπισμός θέσης, οδομετρία, μικροελεγκτής.

Abstract

This diploma thesis deals with issues that concern mobile robots, such as localization of the mobile robot and its navigation in space, specifically motion planning and control of the robot's movement. Particularly omni-directional mobile robots with caster wheels are studied. Furthermore, issues of micro-controller programming, wireless communication, communication with host computer and development of user interface, are presented.

The purpose of the work is to develop a system for controlling an omni-directional robot. The system must include localization methods and trajectory planning, motor driving and communication with sensors. Furthermore, the system must be flexible so that it can be applied to several kind of configurations of omni-directional robot with caster wheels (two wheels, three wheels, etc.) without major changes in programming. The system design should be modular, with individual components that will communicate internally.

Initially the system was designed for an omni-directional robot with three caster wheels. In this dissertation the kinematic model for this robot is developed, but may easily modified to apply to other configurations of omni-directional robot with caster wheels. Based on this kinematic model we develop methodologies for calculating the odometry for localizing the robot.

The most important components are the micro-controller PIC program to drive the wheel motors and the user interface program running on a computer with Windows environment. Even the programs themselves are composed of software modules, including wireless and wired communication, communication with sensors, odometry calculation etc.

Moreover, a simulation of the robot is developed in MatLAB environment for the study of various motion planning algorithms. Goal is to find the best algorithm to be used in the system, which calculates the desired motion profile (position-velocity) to give as input to the system.

Finally the system is applied on two different omni-directional robots (with different configurations) to test its versatility.

Key Words

Omni-directional robot, robot control, path planning, localization, odometry, micro-controller.

Ευχαριστίες / Acknowledgements

Ευχαριστώ την οικογένεια μου, και κυρίως τους γονείς μου, που βρίσκονταν πάντα στο πλάι μου σε όλη τη διάρκεια της ακαδημαϊκής μου πορείας στο Εθνικό Μετσόβιο Πολυτεχνείο, τον καθηγητή μου κ. Κωνσταντίνο Τζαφέστα, που στα χρόνια τα οποία τον γνωρίζω ήταν μια συνεχής πηγή στήριξης και έμπνευσης για μένα, το πανεπιστήμιο Université Pierre et Marie Curie που με φιλοξένησε για να εκπονήσω την διπλωματική μου εργασία στο πλαίσιο του προγράμματος Erasmus, τον υπεύθυνο καθηγητή μου στο UPMC κ. Faïz Ben Amar για την άψογη συνεργασίας μας και όλα τα μέλη του εργαστηρίου Institut des Systèmes Intelligents et de Robotique για την βοήθεια τους.

Κωνσταντίνος Λάντος
Αθήνα, Ιούλιος 2011

I am grateful to my family, especially my parents, who were always by my side throughout my academic course at the National Technical University of Athens, to my professor Mr. Konstantino Tzafesta that in the years that I have known him he was a constant source of support and inspiration for me, to the Université Pierre et Marie Curie who hosted me to conduct research and to work on my diploma thesis in the framework of Erasmus, to the my supervisor professor at UPMC Mr. Faïz Ben Amar for the excellent cooperation and to all members of the Institut des Systèmes Intelligents et de Robotique laboratory for their assistance.

Constantinos Lantos
Athens, July 2011

Περιεχόμενα / Table of Contents

Chapter 1
Introduction

1.1 Purpose

The purpose of this dissertation is the development of a system for odometry localization, trajectory planning and control of an omni-directional robot. The system is developed for omni-directional robots with caster wheels and it must be versatile so that it can be applied easily without many modifications to any configuration (two wheel, three wheel etc.) of omni-directional robots.

Omni-directional robots present a very high interest. They are robots that are able to move in any direction without having to change their orientation. Furthermore, they can change their orientation independently as they move. This feature makes the robots robust and gives them the ability to perform complicated tasks. Also, the omni-directional robots can move in any circumstances that other mobile robots cannot due to their movement restrictions. The localization and trajectory planning of a mobile robot are the basic elements for controlling the robot.

As developed later in the dissertation, the omni-directional ability can be realised by several types of wheels and configurations. Each wheel type and configuration has its advantages and disadvantages. We will deal with caster wheel omni-directional robots. For the purpose of testing and simulation we will develop a three wheel robot. The goal of this dissertation, though, is to develop a complete system and method which can be applied to any configuration of caster wheel omni-directional robot.

Furthermore, the system has to be designed in a modular way, based on individual components that will interact with each other thus giving us the ability to modify a component without having to change the whole system and with only requirement to respect the communication protocol. Therefore, we will be able to implement different robot control and motion planning algorithms without having to alter the other parts of the system.

Along side with the development of the system, a three wheel robot was fabricated to help with the testing of the system. A kinematic model for the robot is introduced and several methods for computing its odometry are presented. These models and methods consider the robot's wheel as parallel joints and so they can be modified easily to suit any caster wheel omni-directional robot. Finally, the system was also applied on a four-wheel robot to test its versatility.

Another part of this diploma thesis is the testing of several path generator algorithms and discovering which produces the best desired position-velocity profile. There are several factors that will be taken in consideration for selecting the best algorithm, like the ability to set maximum velocities and travel most of the movement at those velocities. Another factor to select the best algorithm is how the kinematic model complies and reacts with the velocity profile as input.

The whole process will initiate the reader into many concepts which are relevant to mobile robots, and specifically omni-directional ones. Furthermore, this dissertation will explore aspects of programming and control and it will address problems encountered by mechanical or electronic requirements and restrictions.

1.2 Chapter Analysis

Chapter 2: This chapter is devoted to initiate the reader into the general concepts used in this work. Background and general information on robots, micro-controllers and motors is presented. Extensive descriptions are presented regarding omni-directional robots, their abilities and their use. Wheels that offer omni-directional ability are described and compared, based on recent published reports available in the literature.

Chapter 3: Before designing and implementing the robot we have to define and analyse its kinematic model. Geometrical equations are presented that describe the relations between the robot's wheels and its centre. Using these equations we derive a matrix that is fundamental for the control of the robot. Furthermore, localization through odometry is suggested using two different methods.

Chapter 4: The second stage before the implementation is the simulation of the robot. Using the kinematic model and the odometry methods presented in the previous chapter, we create a simulation of the robot. The simulation is done using MatLAB environment. The basis and algorithm of the simulation is explained, alongside with parts of the code. Next, we use the simulation to test several motion planning algorithms, such as polynomial interpolation functions, potential force fields and position-velocity-time. The results are presented and compared.

Chapter 5: In this chapter the basic electronic hardware, such as the micro-controllers, the motor drivers and the sensors, are described. Also the chapter introduces special hardware, such as slip rings and ZigBee, that was need to be implemented to address problems due to mechanical and electronic restrictions. Photographs, schematics and figures are presented to offer the reader a better understanding of the construction of the robot.

Chapter 6: After presenting all of the aspects of the robot we move to the biggest, and most important, part the implementation of the system. Each program developed for the control of the robot is described, given its algorithm. The possibles configurations of the robots are explained with the corresponding programs to control them. Problems presented in the previous chapter and also problems that occurred in this chapter are addressed and solutions are suggested. In the end, the User Interface program developed for the system is presented and its major parts are explained thoroughly.

Chapter 7: Finally, after implementing all the necessary parts for controlling the robot we perform several experiments to test the system. The two main experiments, that were conducted, tested the versatility of the system, the validity of the odometry methods and the actual velocities of the robot. Two different robots were used to perform the experiments. Screenshots from videos are shown to offer a better view of the experiments.

Chapter 8: In the final chapter there is an overall description of the work, highlighting key points and citing the problems encountered. The conclusion of the dissertation is presented, describing the goals achieved but also the deficiencies and disadvantages of the system. Alongside, there is a reference to future work and to what areas there should be more research to improve.

Chapter 2
Background and General Information

2.1 Omni-directional Robots

As [Camp87] states, mobile robots are called omni-directional when they have full mobility in the plane, which means that they can effortlessly move in any direction without any reorientation. This means that the omni-directional robot is able to move on a straight path from any given point to an other, without having to rotate first. Moreover, the translational movement along a desired path can be combined with a rotation, so that the robot arrives at its destination at the correct angle. In contrast, other robots which use differential driving or car-like robots, have to rotate first towards the goal and then move on a straight line, nonetheless they cannot achieve a desired goal orientation.

The typical types of wheels used in most mobile robots can be classified into the following different types: conventional wheel, centred orientable wheel, off-centred orientable wheel (caster wheel), spherical orthogonal wheel, spherical ball and Swedish wheel. For the mobile robot to have omni-directional characteristics on the plane, only wheels with three degrees of freedom must be employed in mobile robots as stated in [Lee05]. From the above list, the wheels that can be modelled with three degrees of freedom are caster wheel, spherical orthogonal wheel, spherical ball and the Swedish wheel.

The Swedish wheel is the most widely used wheel in omni-directional robots because it can provide omni-directional characteristics with the use of only one motor per wheel. It is a conventional wheel with a series of rollers attached to its circumference. These rollers have an axis of rotation at 45° to the plane of the wheel in a plane parallel to the axis of rotation of the wheel. By using left and right-handed rollers at adjacent wheels, the vehicle is stable and can be made to move in any direction and turn by varying the direction and speed of each wheel. [Muir87] developed Uranus omni-directional robot with four Swedish wheels.

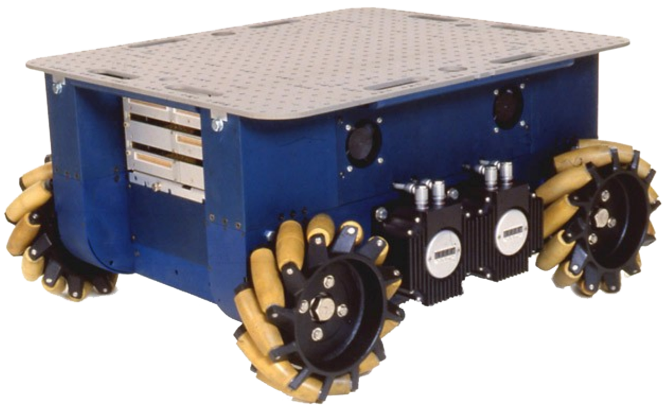


Fig 2.1 URANUS robot [Muir87]

The spherical orthogonal wheel, introduced by [Pin94] and [Pois01], is an idea that emanates from the concept that an ideal wheel for an omni-directional robot is a sphere nevertheless, a sphere cannot be powered without losing one of its three degrees of mobility. So the spherical orthogonal wheel is formed with two truncated spheres which are intermechanically dependent. These wheels were used by [Mour06] to develop an omni-directional robot named ROMNI that uses three orthogonal wheels.

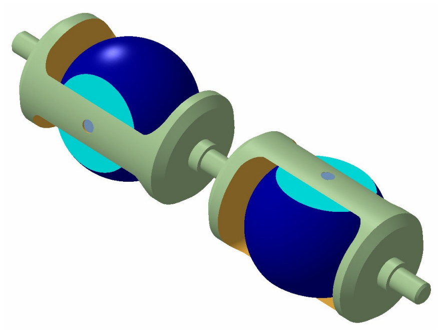


Fig 2.2 Spherical Orthogonal Wheel [Mour06]

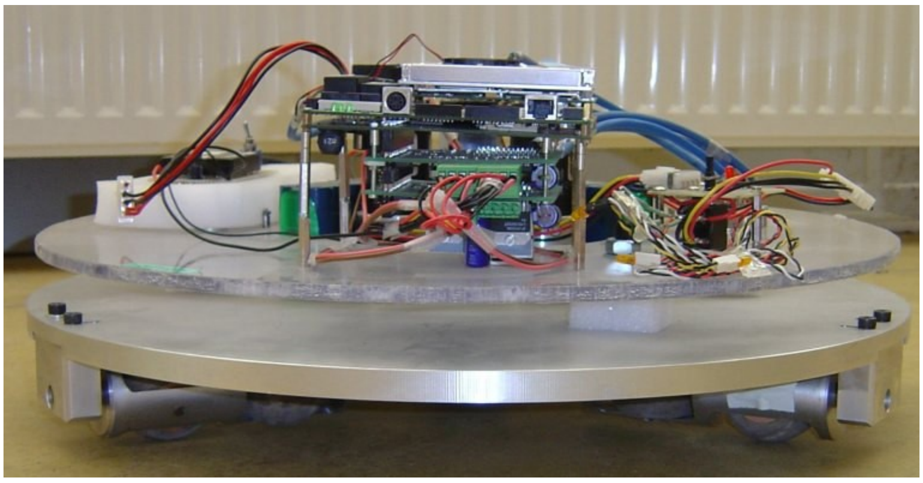


Fig 2.3 ROMNI robot [Mour06]

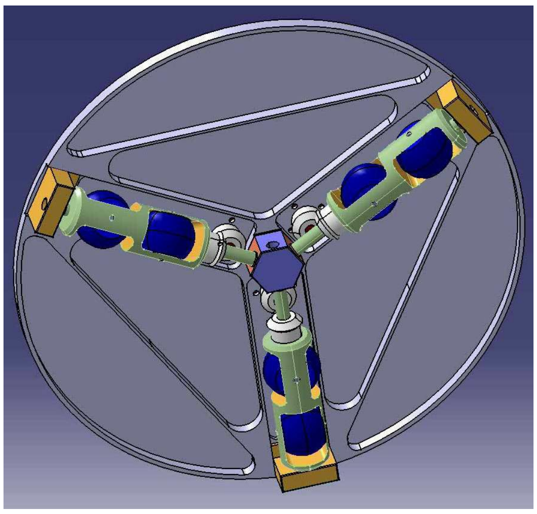


Fig 2.4 Layout of the three axles (CAD design) ROMNI robot [Mour06]

[West95] has managed to develop a spherical ball wheel that consists of a ball that supports a vehicle chassis upon a spherical tyre such that the chassis may roll in any direction on the floor. There is also form closure around the ball, provided by the ball wheel mechanism and contact with the floor surface. The ball wheel mechanism rotates the ball around a desired axis and thus moving the vehicle.

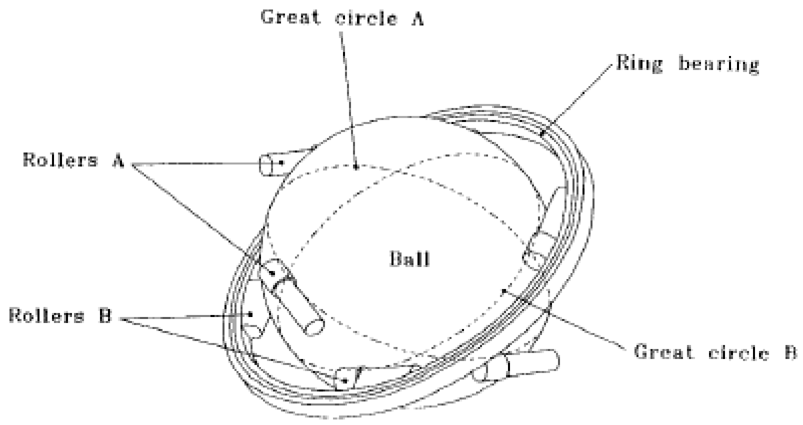


Fig 2.5 Spherical ball wheel [West95]

However, both the Swedish wheel and most types of "omni-directional wheels" are notorious for their sensitivity to road conditions, which renders their operational performances more or less limited compared to conventional wheels. On the other hand, the active caster wheel is not sensitive to road conditions and is also able to overcome steps encountered in uneven floors by using the active driving wheel. Recent developments are the six-wheeled omni-directional mobile robot of [Moore00] and ODIS (Omni-Directional Inspection System) of [Berk05] with the so-called "smart wheel" active caster wheel module. Also, [Wada00] developed the mobile robot with two caster wheels and one rotational actuator and [Ush03] developed an omni-directional vehicle with two-wheeled casters.

2.2 Micro-controllers

An essential component of the robot is the micro-controllers which are used for controlling the servos, sending commands from the computer or communicating with sensors. A micro-controller is actually a small computer on a single integrated circuit that includes a processor, memory and input/output communication. Micro-controllers are used mainly in embedded systems because they combine all the components of a computer in a single chip, rather than micro-processors which need external memory and other peripherals. The two most known types of micro-controllers are the AVR and the PIC.

Micro-controllers usually contain several general purpose input/output pins (GPIO). GPIO pins are software configurable to either an input or an output state. When GPIO pins are configured to an input state, they are often used to read sensors or external signals. Configured to the output state, GPIO pins can drive external devices such as LEDs or motors. Many embedded systems need to read sensors that produce analog signals. This is the purpose of the analog-to-digital converter (ADC). Since processors are built to interpret and process digital data they are not able to do anything with the analog signals that may be sent by a sensor. So the analog-to-digital converter is used to convert the incoming data into a form that the processor can recognise. Furthermore, micro-controllers have serial/USB communication interface that allows them to communicate with a computer or other device.

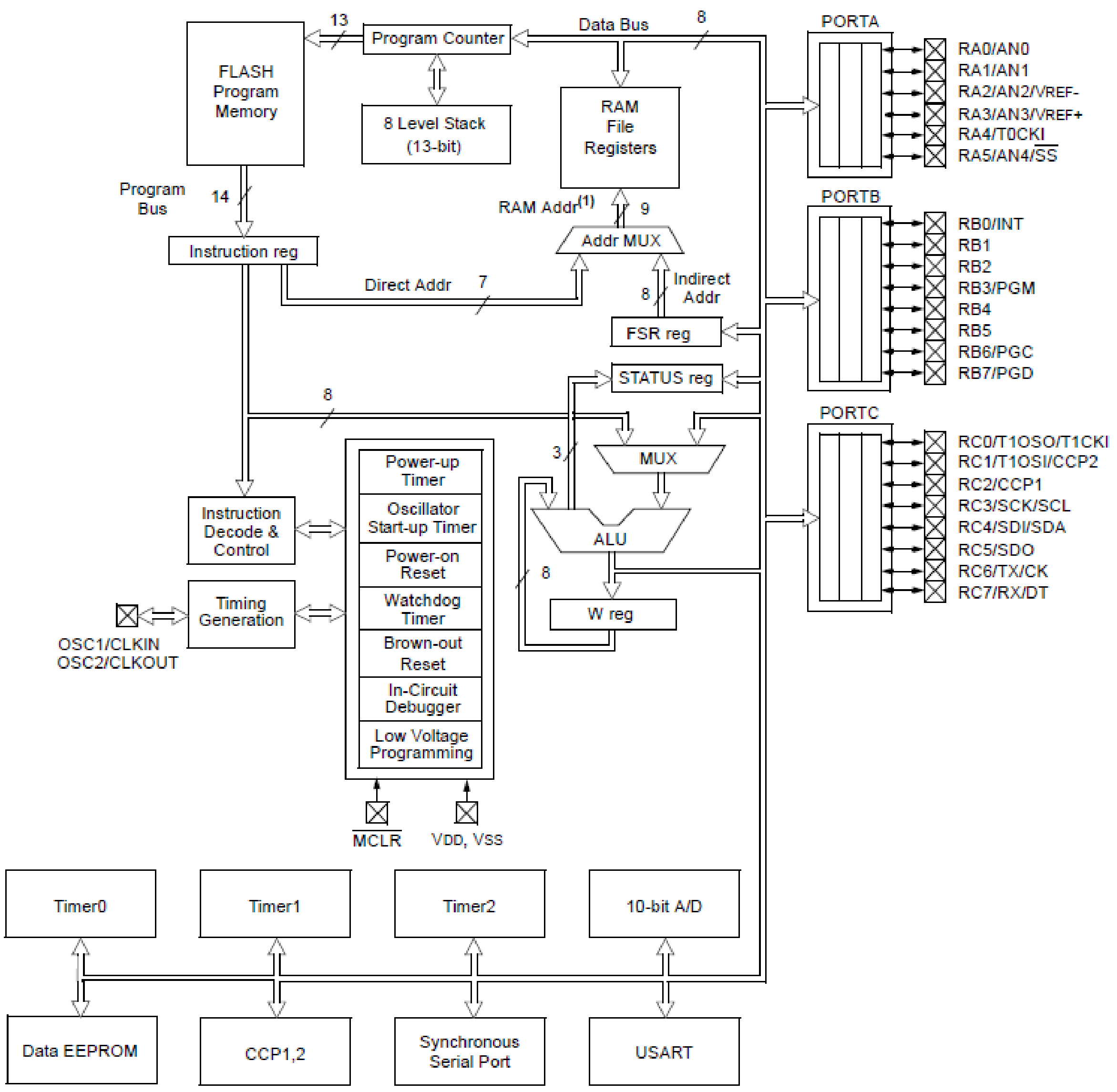


Fig 2.6 16F87X PIC Architecture [Pekm09]

2.3 Stepper Motor

Stepper motors have multiple "toothed" electromagnets arranged around a central gear-shaped piece of iron. The electromagnets are energised by an external control circuit, such as a microcontroller or stepper driver. To make the motor shaft turn, first one electromagnet is given power, which makes the gear's teeth magnetically attracted to the electromagnet's teeth. When the gear's teeth are thus aligned to the first electromagnet, they are slightly offset from the next electromagnet. Therefore, when the next electromagnet is turned on and the first is turned off, the gear rotates slightly to align with the next one, and from there the process is repeated. Each of those slight rotations is called a "step", with an integer number of steps making a full rotation. In that way, the motor can be turned by a precise angle. As motor speed increases, torque decreases, but the torque curve may be extended by using current limiting drivers and increasing the driving voltage. This dissertation will introduce and examine two-phase stepper motors.

2.3.1 Two-Phase Unipolar motors

A unipolar stepper motor has two windings per phase, one for each direction of magnetic field. Typically, given a phase, one end of each winding is made common: giving three leads per phase and six leads for a typical two-phase motor. Since in this arrangement a magnetic pole can be reversed without switching the direction of current, the commutation circuit can be made very simple for each winding. A microcontroller or stepper motor controller can be used to activate the drive transistors in the right order.

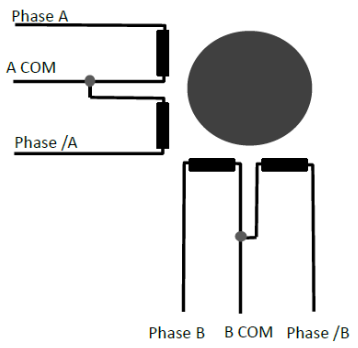


Fig 2.7 Two-Phase Unipolar stepper motor [Sopr57] [Tech23]

2.3.2 Two-Phase Bipolar motor

Bipolar motors have a single winding per phase. There are two leads per phase, neither of which is common. The current in a winding needs to be reversed in order to reverse a magnetic pole, so the driving circuit must be more complicated. Because windings are better utilised, they are more powerful than a unipolar motor of the same weight. This is due to the physical space occupied by the windings. A unipolar motor has twice the amount of wire in the same space, but only half used at any point in time. Though a bipolar motor is more complicated to drive, the abundance of driver chips indicates that it is much less difficult to achieve.

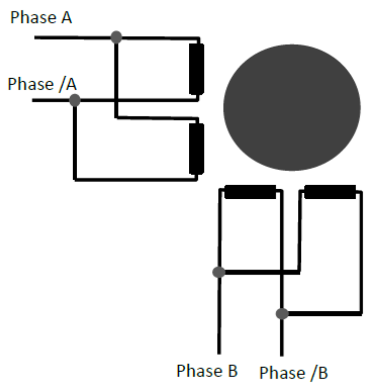


Fig 2.8 Two-Phase Bipolar motor [Sopr57] [Tech23]

Chapter 3
Kinematic Modeling and Odometry Localization

3.1 Kinematic Model

The kinematic model explained here is for a robot which bears three caster wheels, but can also be modified easily to any robot using the same methods. The robot has a circular shape and the wheels are placed in a triangle. In this way we can use the kinematic model described in [Lee05]. The robot can be visualised as a triangle robot with the three caster wheels and a circular plate on top.

To define the kinematic model of the robot, we assume that the motion of the mobile robot is constrained to the plane and there is no sliding and skidding friction, but the rotation of wheel about the axis vertical to the ground is allowed. The robot is able to move only in a two axis coordinate system (X_G,Y_G) and so we have only three variables (x_c,y_c,ϕ_c) which represent the unit vectors of the body frame fixed to the body of the mobile robot's centre, respectively, and ϕ_c is the orientation of the robot (angle between the local and the global X axis). The wheels are also described by three variables (θ_i,ϕ_i,η_i) where i denotes the number of the wheel (1,2,3). θ_i denotes the rotating angle of the wheel and ϕ_i denotes the steering angle between steering link and the local X axis. η_i denotes the angular displacement of the wheel relative to the global X axis of the reference frame. It can be derived that η_i is depended on ϕ_i and ϕ_c, as could be expected since, at each wheel, the motors control only the θ_i, ϕ_i joints. A joint is actuated (active) only when it is controlled by a motor, so that we can define its value. A passive joint is a joint that moves freely without being controlled, but sometimes can depend on other variables that are controllable.

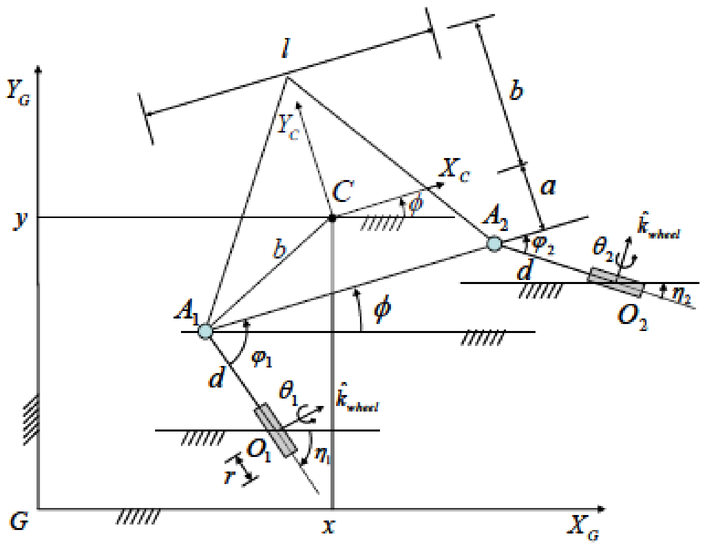


Fig 3.1 Geometrical representation of the robot [Lee05]

We define the output velocity of the robot's centre as $u_c=\begin{bmatrix} v_{cx} \\ v_{cy} \\ \omega \end{bmatrix}$ where v_{cx}, v_{cy} is the translational velocity and ω the angular velocity about the vertical axis. The velocities v_{cx}, v_{cy} are expressed in the local coordinate system, but the trajectory planning algorithms calculate the desired velocities in global coordinate system. So the desired velocity vector must be transformed using the rotation matrix G→C:

$$\begin{bmatrix} v_{cx} \\ v_{cy} \\ \omega \end{bmatrix} = R_G^C \begin{bmatrix} \dot{x}_{cd} \\ \dot{y}_{cd} \\ \dot{\phi}_{cd} \end{bmatrix} \text{, where } R_G^C=(R_C^G)^{-1}=(R_C^G)^T=\begin{bmatrix} \cos\phi_c & \sin\phi_c & 0 \\ -\sin\phi_c & \cos\phi_c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The mobile robot can be considered as a parallel mechanism, so each wheel contributes parallel to the velocity of the robot's centre. Using the procedure described in [Lee05] we derived the following equations:

$$\begin{bmatrix} v_{cx} \\ v_{cy} \\ \omega \end{bmatrix} = \begin{bmatrix} -d\sin\phi_1 - a & r\cos\phi_1 & -a \\ -d\cos\phi_1 + \frac{l}{2} & -r\sin\phi_1 & \frac{l}{2} \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\eta}_1 \\ \dot{\theta}_1 \\ \dot{\phi}_1 \end{bmatrix}$$

$$\begin{bmatrix} v_{cx} \\ v_{cy} \\ \omega \end{bmatrix} = \begin{bmatrix} -d\sin\phi_2 - a & r\cos\phi_2 & -a \\ -d\cos\phi_2 - \frac{l}{2} & -r\sin\phi_2 & -\frac{l}{2} \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\eta}_2 \\ \dot{\theta}_2 \\ \dot{\phi}_2 \end{bmatrix}$$

$$\begin{bmatrix} v_{cx} \\ v_{cy} \\ \omega \end{bmatrix} = \begin{bmatrix} -d\sin\phi_3 + b & r\cos\phi_3 & b \\ -d\cos\phi_3 & -r\sin\phi_3 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\eta}_3 \\ \dot{\theta}_3 \\ \dot{\phi}_3 \end{bmatrix}$$

where α, b, d, l, r are the robot's characteristics and are shown on the figure above.

Taking the inverse of the above equations we have:

$$\begin{bmatrix} \dot{\eta}_1 \\ \dot{\theta}_1 \\ \dot{\phi}_1 \end{bmatrix} = \frac{1}{dr} \begin{bmatrix} -r\sin\phi_1 & -r\cos\phi_1 & \frac{l}{2}r\cos\phi_1 - ar\sin\phi_1 \\ d\cos\phi_1 & -d\sin\phi_1 & \frac{l}{2}d\sin\phi_1 + ad\cos\phi_1 \\ r\sin\phi_1 & r\cos\phi_1 & dr + ar\sin\phi_1 - \frac{l}{2}r\cos\phi_1 \end{bmatrix} \begin{bmatrix} v_{cx} \\ v_{cy} \\ \omega \end{bmatrix}$$

$$\begin{bmatrix}\dot{\eta}_2\\ \dot{\theta}_2\\ \dot{\phi}_2\end{bmatrix}=\frac{1}{dr}\begin{bmatrix}-r\sin\phi_2 & -r\cos\phi_2 & \frac{l}{2}r\cos\phi_2-ar\sin\phi_2\\ d\cos\phi_2 & -d\sin\phi_2 & -\frac{l}{2}d\sin\phi_2+ad\cos\phi_2\\ r\sin\phi_2 & r\cos\phi_2 & dr+ar\sin\phi_2+\frac{l}{2}r\cos\phi_2\end{bmatrix}\begin{bmatrix}v_{cx}\\ v_{cy}\\ \omega\end{bmatrix}$$

$$\begin{bmatrix}\dot{\eta}_3\\ \dot{\theta}_3\\ \dot{\phi}_3\end{bmatrix}=\frac{1}{dr}\begin{bmatrix}-r\sin\phi_3 & -r\cos\phi_3 & -br\cos\phi_3-ar\sin\phi_3\\ d\cos\phi_3 & -d\sin\phi_3 & bd\cos\phi_3\\ r\sin\phi_3 & r\cos\phi_3 & dr-br\sin\phi_3\end{bmatrix}\begin{bmatrix}v_{cx}\\ v_{cy}\\ \omega\end{bmatrix}$$

$\dot{\eta}_1, \dot{\eta}_2, \dot{\eta}_3$ are passive joints, they are used only to complete the kinematic model and they can be eliminated with $\eta_i=\varphi_i-\varphi_c$ (derived by Fig 3.1). Using the inverse equations for the active joints $\dot{\theta}_1, \dot{\phi}_1, \dot{\theta}_2, \dot{\phi}_2, \dot{\theta}_3, \dot{\phi}_3$ we form the K matrix, which acts like an inverse Jacobian matrix:

$$\begin{bmatrix}\dot{\theta}_1\\ \dot{\phi}_1\\ \dot{\theta}_2\\ \dot{\phi}_2\\ \dot{\theta}_3\\ \dot{\phi}_3\end{bmatrix}=K(\phi_i)\begin{bmatrix}v_{cx}\\ v_{cy}\\ \omega\end{bmatrix}$$

$$K(\phi_i)=\frac{1}{dr}\begin{bmatrix}d\cos\phi_1 & -d\sin\phi_1 & \frac{l}{2}d\sin\phi_1+ad\cos\phi_1\\ r\sin\phi_1 & r\cos\phi_1 & dr+ar\sin\phi_1-\frac{l}{2}r\cos\phi_1\\ d\cos\phi_2 & -d\sin\phi_2 & -\frac{l}{2}d\sin\phi_2+ad\cos\phi_2\\ r\sin\phi_2 & r\cos\phi_2 & dr+ar\sin\phi_2+\frac{l}{2}r\cos\phi_2\\ d\cos\phi_3 & -d\sin\phi_3 & bd\cos\phi_3\\ r\sin\phi_3 & r\cos\phi_3 & dr-br\sin\phi_3\end{bmatrix}$$

3.2 Odometry

To be able to control the motion of the robot we have to know its posture (position and orientation with respect to the global reference frame) at each time step. One basic method to perform this robot localization task is calculating its odometry. Odometry is the use of data from sensors and kinematic model to estimate change in position over time. Odometry is used by some robots, to estimate (not determine) their position relative to a starting location. This method is sensitive to errors due to the integration of velocity measurements over time to give position estimates.

The posture of the mobile robot can be described in terms of the two coordinates x and y of the origin C of the moving frame and the orientation angle ϕ of the moving frame, both with respect to the global frame with origin at G.

3.2.1 1st Method

The 1st method is based on the algorithm described in [Jung08], according to which we have the following steps to compute the posture of the robot:

Step 1: Setting the initial positions

$$x_c[0]=x_{c0} \;,\; y_c[0]=y_{c0} \;,\; \phi_c[0]=\phi_{c0}$$
$$\theta_i=0(stop) \;,\; \phi_i[0]=sensor\,reading_i$$
$$\eta_i[0]=\phi_i[0]-\phi_c[0]$$
$$\left.\begin{cases} x_{O1}[0]=x_c[0]-b\cos\left(\frac{\pi}{6}+\phi_c[0]\right)+d\cos(\eta_1[0]) \\ y_{O1}[0]=y_c[0]-b\sin\left(\frac{\pi}{6}+\phi_c[0]\right)-d\sin(\eta_1[0]) \end{cases}\right\} position\,of\,caster\,wheel\,1\,(O_1)$$

Step 2: Get sensor readings for (real) orientation angles of each wheel and compute pseudo-inverse K^+

$$\phi_i[n]=sensor\,reading_i$$
$$K^+(\phi_i[n])=[K^T(\phi_i[n])K(\phi_i[n])]^{-1}K^T(\phi_i[n])$$

Step 3: Calculate (real) angular velocity and orientation angle of C

$$\begin{bmatrix} - \\ - \\ \omega \end{bmatrix}=K^+(\phi_i[n])\cdot\begin{bmatrix} \dot{\theta}_1 \\ \dot{\phi}_1 \\ \dot{\theta}_2 \\ \dot{\phi}_2 \\ \dot{\theta}_3 \\ \dot{\phi}_3 \end{bmatrix}$$
$$\phi_c[n]=\phi_c[n-1]+\omega\Delta t$$

Step 4: Update position of caster wheel 1

$$\eta_1[n]=\phi_1[n]-\phi_c[n]$$
$$x_{O1}[n+1]=x_{O1}[n]+r\,\dot{\theta}_1[n]\,\Delta t\cos(\eta_1[n])$$
$$y_{O1}[n+1]=y_{O1}[n]-r\,\dot{\theta}_1[n]\,\Delta t\sin(\eta_1[n])$$

Step 5: Calculate the position of steering point A_1

$$x_1[n]=x_{O1}[n]-d\cos(\eta_1)$$
$$y_1[n]=y_{O1}[n]+d\sin(\eta_1)$$

Step 6: Obtain the posture of the mobile robot C

$$x_c[n]=x_1[n]+b\cos(\frac{\pi}{6}+\phi_c[n])$$
$$y_c[n]=y_1[n]+b\sin(\frac{\pi}{6}+\phi_c[n])$$

The odometry information of the omni-directional mobile robot with active caster wheels can be obtained by using this procedure at every encoder sampling time iteratively. Our method is slightly different from the method described in [Jung08] because we compute ϕ_c with the angular velocity each time step rather than the atan2 function. The atan2 function has the disadvantage of being limited inside [-π, π] thus not providing the ability to define goals with ϕ_c outside of those borders.

Furthermore, it is notable that the position of the robot's centre is calculated by the use of only one wheel. Any sliding of the wheel will affect the calculation of the robot's centre and lead to errors. Therefore, it was considered preferable to calculate the position of the robot's centre with the use of the other wheels, using the same method and yielding three different calculations of the same point. By taking the mean value, the possibility of errors is minimised.

3.2.2 2nd Method

A 2nd method emerged when studying and analysing the 1st method. Since we were calculating the K^+ for computing the angular velocity, another approach could be to compute also the linear velocities and integrate them to find the position of the robot. Thus we have:

Step 1: Setting the initial positions

$$x_c[0]=x_{c0}\ ,\ y_c[0]=y_{c0}\ ,\ \phi_c[0]=\phi_{c0}$$
$$\phi_i[0]=sensor\ reading_i$$

Step 2: Get sensor readings for (real) orientation angles of each wheel and compute pseudo-inverse K^+

$$\phi_i[n]=sensor\ reading_i$$
$$K^+(\phi_i[n])=[K^T(\phi_i[n])K(\phi_i[n])]^{-1}K^T(\phi_i[n])$$

Step 3: Calculate (real) angular and linear velocity of C (local frame) and convert them to Global frame coordinates

$$\begin{bmatrix} v_{cx} \\ v_{cy} \\ \omega \end{bmatrix} = K^{+}(\phi_i[n]) \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\phi}_1 \\ \dot{\theta}_2 \\ \dot{\phi}_2 \\ \dot{\theta}_3 \\ \dot{\phi}_3 \end{bmatrix}$$

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = R_C^G \begin{bmatrix} v_{cx} \\ v_{cy} \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\phi_c[n]) & -\sin(\phi_c[n]) & 0 \\ \sin(\phi_c[n]) & \cos(\phi_c[n]) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{cx} \\ v_{cy} \\ \omega \end{bmatrix}$$

Step 4: Obtain the posture of the mobile robot C

$$x_c[n+1] = x_c[n] + v_x \Delta t$$
$$y_c[n+1] = y_c[n] + v_y \Delta t$$
$$\phi_c[n+1] = \phi_c[n] + \omega \Delta t$$

This method does not involve the computation of intermediate points, like the positions of the wheels or the joints, but only their orientation and velocities for the specific time step. This can give a better estimation of the robot's position as it calculates a global velocity for the robot's center, thus taking the robot as a whole and calculating how each wheel contributes to its movement. In contrast, the 1st method calculates 3 individual positions based on each wheel differently, meaning how the robot would move if only the specific wheel existed. However, both methods rely on $\dot{\theta}_i$, as kinematic constraints to be respected which is a disadvantage.

Chapter 4
Simulation

Before developing the robot and implementing all these equations to a program, it was decided to make a simulation of the robot, so as to test the reliability of the chosen analysis. Also, the simulation would provide the opportunity to try out several trajectory planning algorithms and to determine which reacts better with the kinematic model. There wasn't any need to try different methods of calculating odometry because the representation and animation of the robot is constructed only by odometry data, so it wouldn't make any difference. Our goal was to find out which trajectory planning algorithm suited best on following a given path and minimizing errors between the odometry data and the path. The simulation was made with the use of MatLAB.

4.1 The basis of the simulation

First, our goal was to create a basis for the simulation that would consist of initial conditions, calculation of the joint's vector and the robot's posture with numerical integration and animation of the robot's path. The initial conditions were the dimensions of the robot, the initial and the final position of the robot and initial joint positions. Between the initial conditions and the numerical integration we could add the desired motion planning algorithm so we could see how the robot would comply with the desired path.

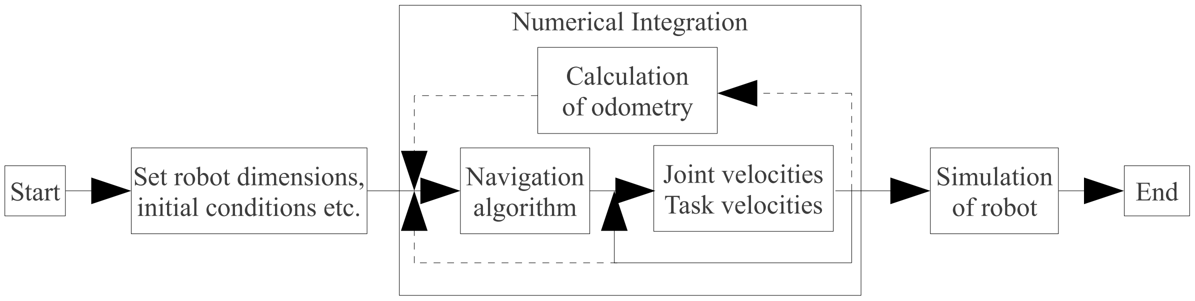


Fig 4.1 MatLAB simulation algorithm

The numerical integration computes the joint velocities and the posture of the robot for the entire time frame. It is done with the use of the function ode45:

```
    [T,Q] = ode45('omni_inv',t,init);
```

where $T(t)=t$, $Q(t)=[\theta_1(t)\ \ \phi_1(t)\ \ \theta_2(t)\ \ \phi_2(t)\ \ \theta_3(t)\ \ \phi_3(t)\ \vdots\ x_c(t)\ \ y_c(t)\ \ \phi_c(t)]^T$, t is time frame and init are the initial positions of the joints.

`omni_inv` is the function whereby the trajectory planning algorithms (if necessary), the odometry data and the joint and task velocities are calculated :

```
%% ***** Calculate Odometry data *****

…

%% Calculation of O1,O2,O3
od_x(1) = od_x0(1)-d*cos(od_h(1));
od_y(1) = od_y0(1)+d*sin(od_h(1));
od_x(2) = od_x0(2)-d*cos(od_h(2));
od_y(2) = od_y0(2)+d*sin(od_h(2));
od_x(3) = od_x0(3)-d*cos(od_h(3));
od_y(3) = od_y0(3)+d*sin(od_h(3));

%% Calculation of robot's center
xc = (od_x(1)+od_x(2)+od_x(3)) / 3;
yc = (od_y(1)+od_y(2)+od_y(3)) / 3;
fc = fc+dp(3)*dt;

%% ***** Calculate  Control Signal (motion planning
%% Algorithm) *****
%% i.e. Polynomial interpolation functions
u(1,1)=5*ax(1)*t^4+4*ax(2)*t^3+3*ax(3)*t^2;
u(2,1)=5*ay(1)*t^4+4*ay(2)*t^3+3*ay(3)*t^2;
u(3,1)=5*af(1)*t^4+4*af(2)*t^3+3*af(3)*t^2;

%% The u (control) vector which is the desired task
%% velocities

%% ***** Inverse Kinematics *****

c1 = cos(f(1)); s1 = sin(f(1));
c2 = cos(f(2)); s2 = sin(f(2));
c3 = cos(f(3)); s3 = sin(f(3));

K(1,1) = d*c1;
K(1,2) = -d*s1;
K(1,3) = 1/2*d*s1+a*d*c1;
…

dq=K*R'*u; % joint velocities

%% ***** Forward Kinematics *****

K_psinv = (K'*K)^-1*K';

dp = R*K_psinv*dq; % task velocities
```

R is the rotation matrix between the global frame and the local frame of the robot and u is the control signal (desired velocities). In most cases, the trajectory planning algorithm must be calculated inside the numerical integration for each time step because it needs the localisation data and it also needs to produce the desired velocities for the specific time step.

As derived from above, the simulation of the robot is the numerical integration of the robot's velocities calculated based on the inverse K matrix and the joint velocities. Of course this is not accurate as we do not take into consideration any slip of the wheels, friction and other distortions that will exist in the real robot.

The representation of the robot is shown in the figure below:

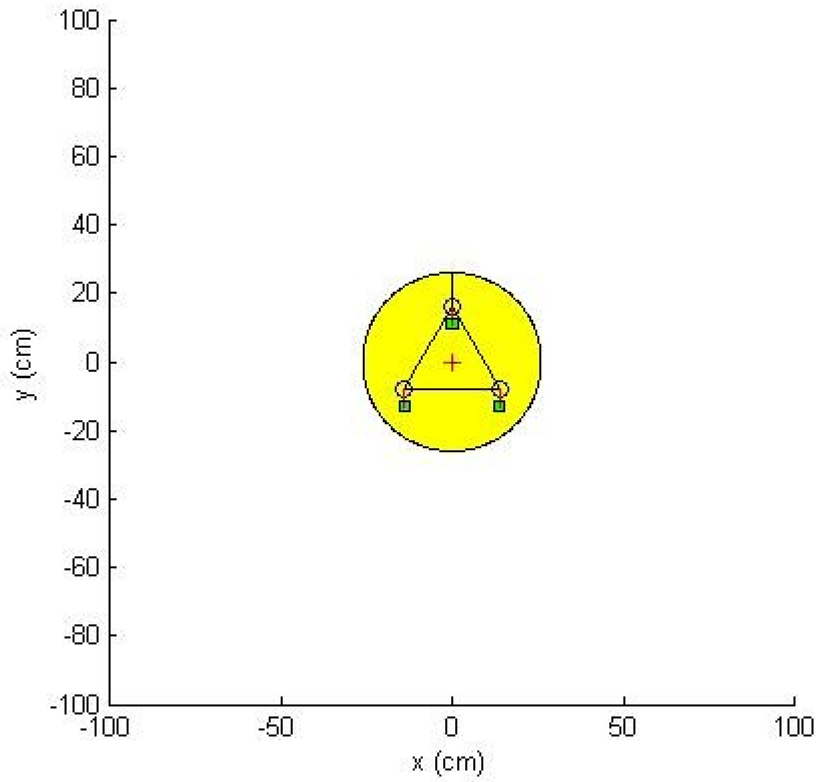


Fig 4.2 Representation of robot in simulation

The yellow circle represents the robot's surface. The green rectangles are the wheel's, the red lines and the small circles are the joints of the wheels with the robot's chassis. The triangle represents the triangle of the kinematic model. The cross is the global point (0,0). The black line on top of the robot exists only to denote the orientation of the robot.

4.2 Motion Planning Algorithms

4.2.1 Polynomial interpolation functions

The first idea was to use a polynomial function for trajectory planning and to calculate the desired velocities. We chose a 5th degree polynomial so that we could meet all of the initial conditions, and obtain continuous velocity and acceleration profiles.

$$x(t)=a_5t^5+a_4t^4+a_3t^3+a_2t^2+a_1t+a_0 \quad 0\leq t\leq T$$
$$\Rightarrow \dot{x}(t)=5a_5t^4+4a_4t^3+3a_3t^2+2a_2t+a_1$$
$$\ddot{x}(t)=20a_5t^3+12a_4t^2+6a_3t+2a_2$$
$$x(0)=x_0 \ , \ x(T)=x_d \ , \ \dot{x}(0)=0 \ , \ \dot{x}(T)=0 \ , \ \ddot{x}(0)=0 \ , \ \ddot{x}(T)=0$$
$$\Rightarrow a_0=x_0 \ , \ a_1=0 \ , \ a_2=0 \ , \ a_3=\frac{10(x_d-x_0)}{T^3} \ , \ a_4=\frac{15(x_0-x_d)}{T^4} \ , \ a_5=\frac{6(x_d-x_0)}{T^5}$$

This was calculated for y(t) and ϕ(t) with y_0, y_d and ϕ_0, ϕ_d respectively. So having the $\dot{x}(t),\dot{y}(t),\dot{\phi}(t)$, we insert them in the numeric integration that produces the joint velocities. Using the joint velocities we calculate the robot posture at each time step.

Example with $x_0=20, y_0=-40, \phi_0=-\frac{\pi}{2}$, $x_d=-30, y_d=20, \phi_d=\pi$, $T=30\,s$:

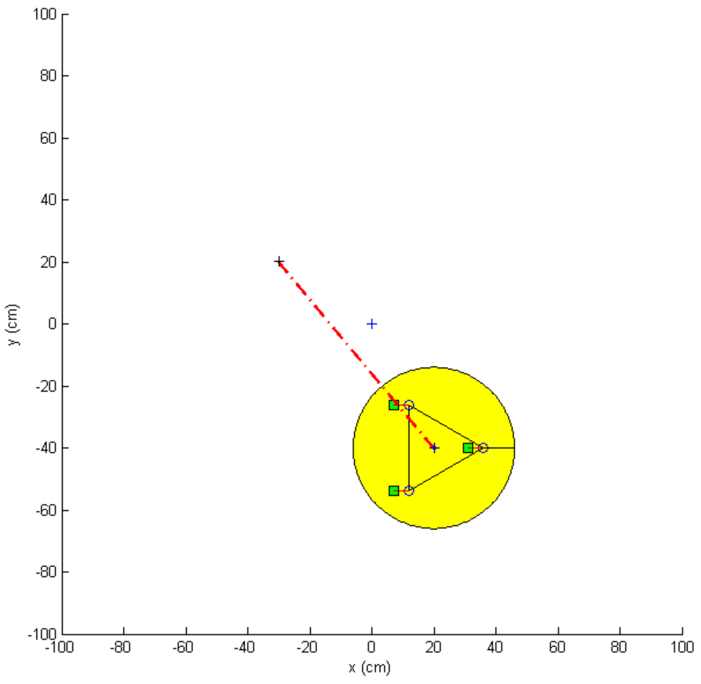


Fig 4.3 Robot's snapshot at t=0s

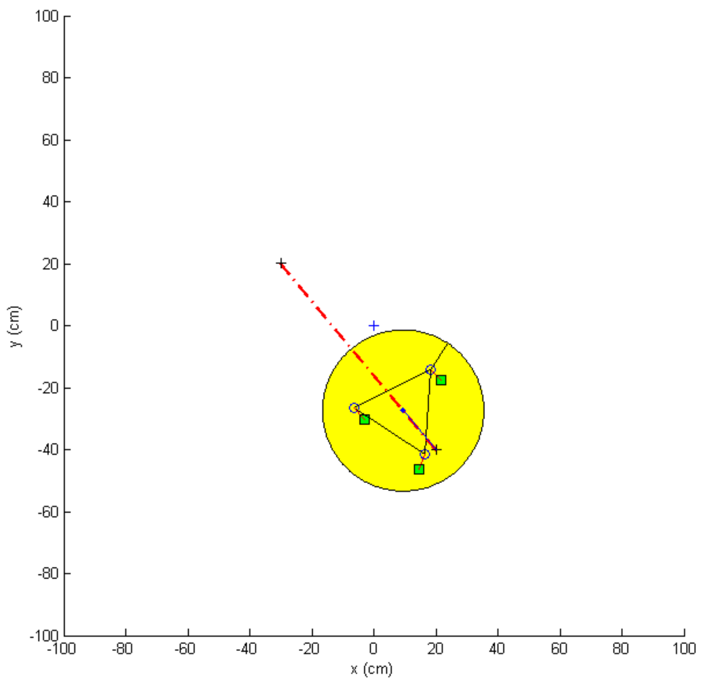


Fig 4.4 Robot's snapshot at t=10s

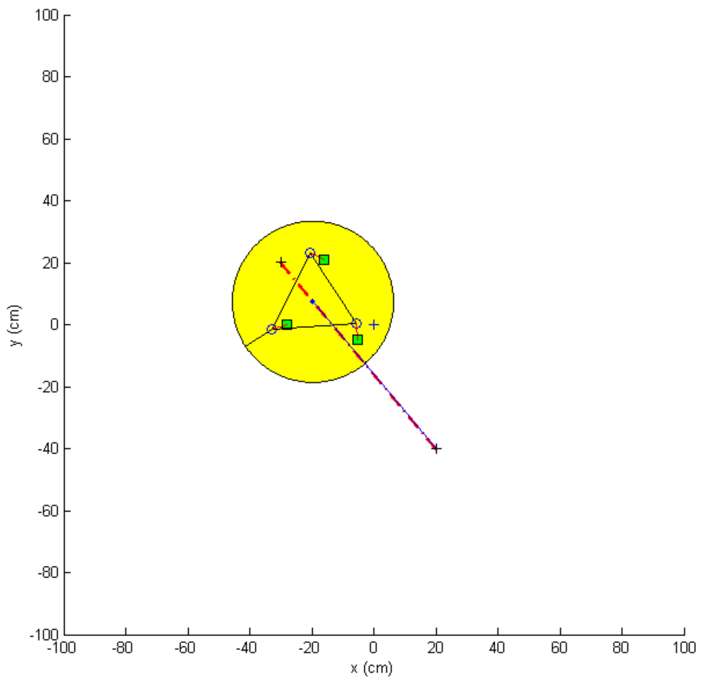


Fig 4.5 Robot's snapshot at t=20s

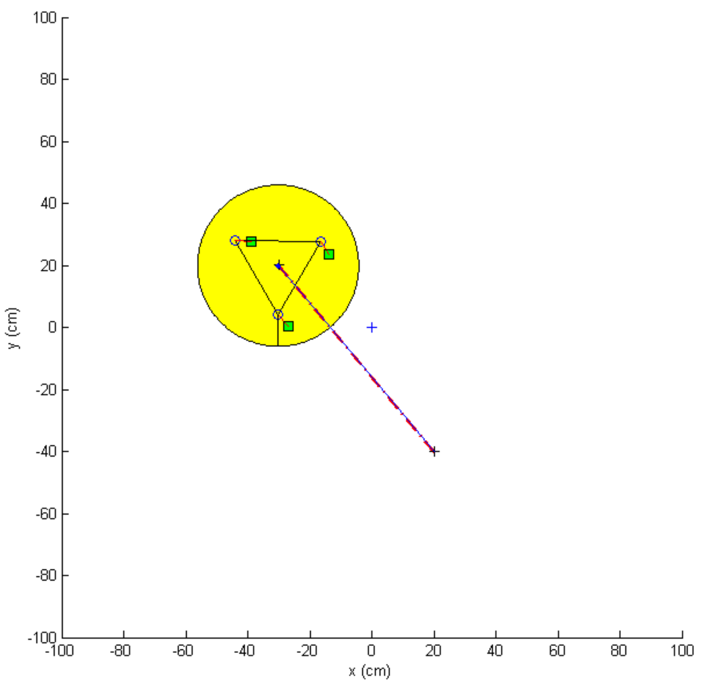


Fig 4.6 Robot's snapshot at t=30s

The above figures are snapshots of the robot's movement each ten seconds. The red dotted line denotes the desire path, the black crosses are the start and goal point and the blue cross is the point (0,0). For more detail we represent the robot's position in respect to time to the figures below. There is also a red line, which denotes the desired position as calculated by the motion planning algorithm.

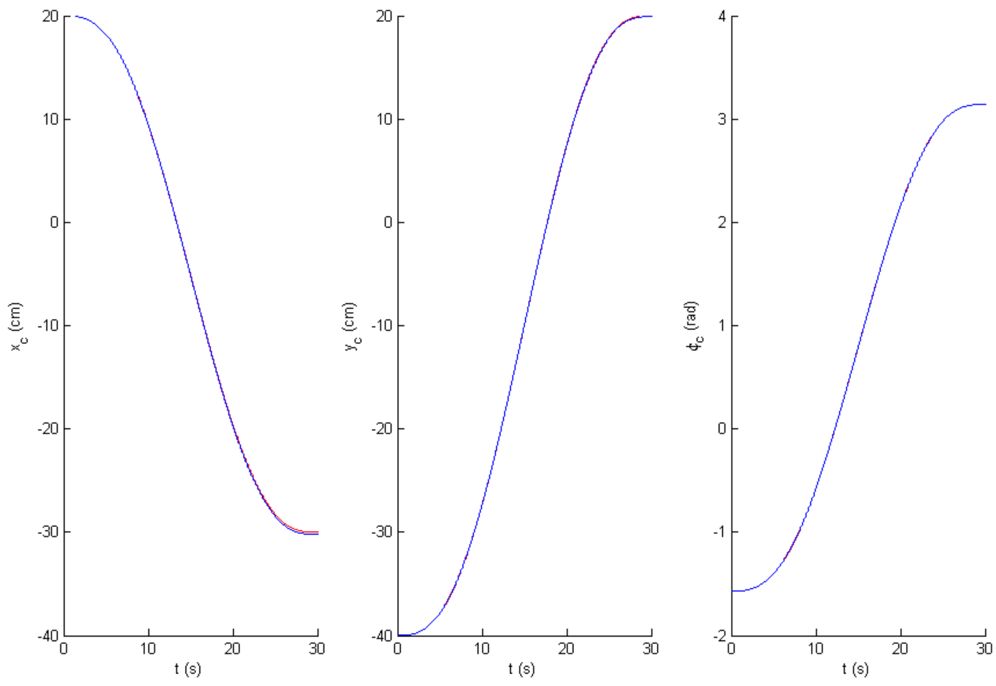


Fig 4.7 Robot's position

The Joint Velocities produced by the Kinematic model are shown below.

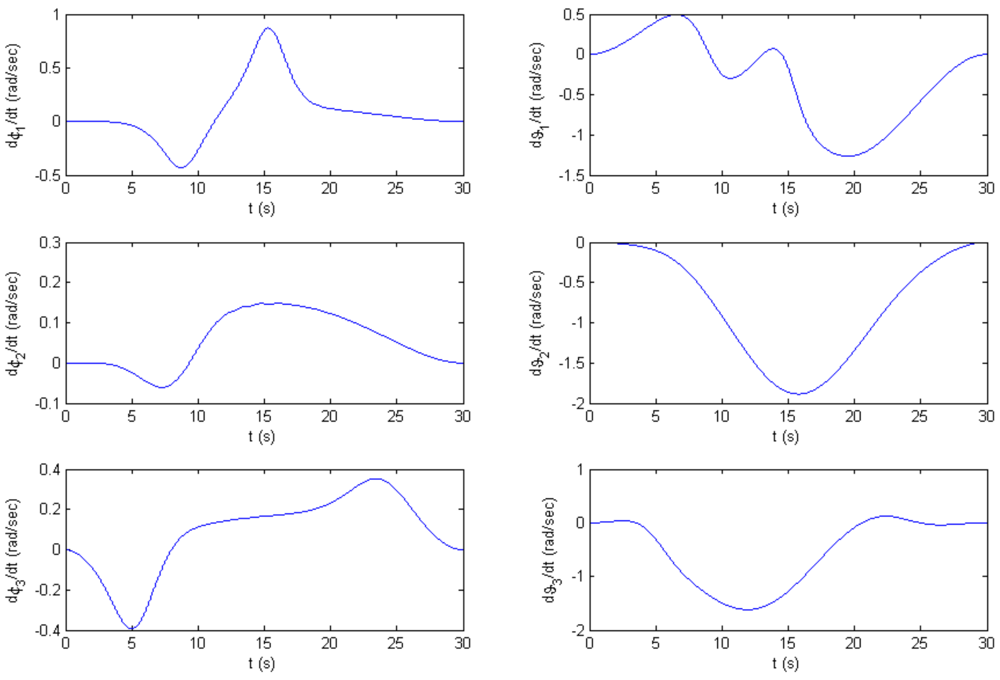


Fig 4.8 Joint Velocities

The robot's velocities calculated by the inverse K matrix and the joint velocities are shown below. Also there is a red line which denotes the desired velocities calculated by the motion planning algorithm.

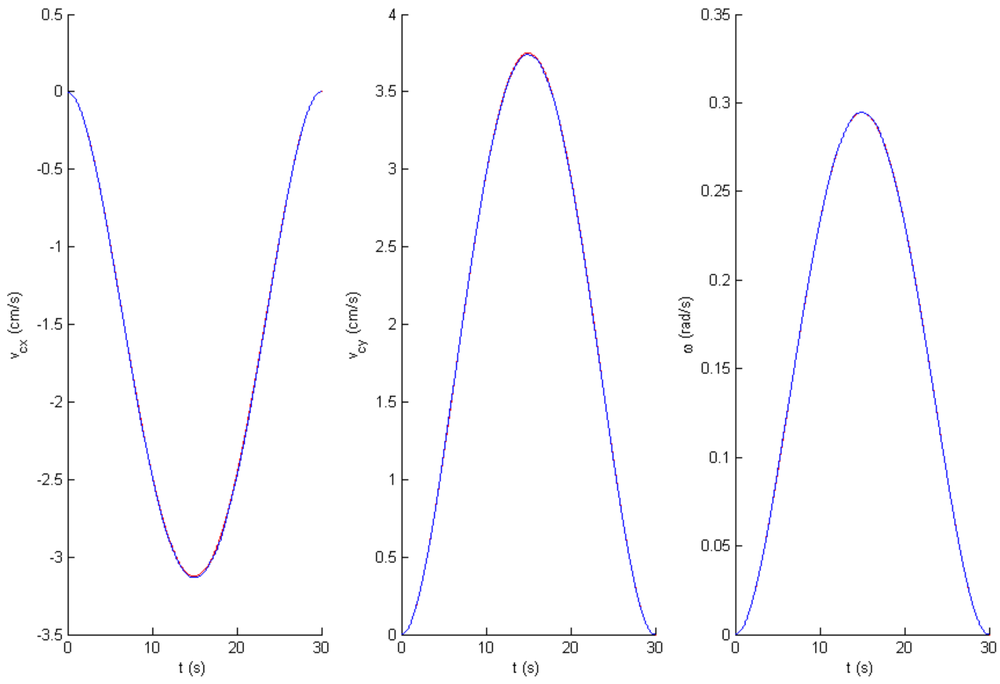


Fig 4.9 Robot's velocities

We notice that the robot followed the desired path without any significant errors. The task velocities (blue) were identical with the desired task velocities (red). But the control applied here is an open-loop system without the consideration of odometry or any other localisation technique. This is an offline trajectory generation algorithm which in most cases of mobile robots, is not applicable. In this case we suppose that the wheels were moving with the speed applied to them without any slipping and also the angular velocity of each wheel is respected without any errors. Of course we cannot suppose that this will also be the case in the real robot. The rotations of the wheels related to the robots frame are not guaranteed and that is why we will measure it with an absolute sensor, as the only thing we suppose is true is the linear velocity of each wheel.

We can alter the above algorithm to compute odometry data at each time step, compare them with the desired path and velocities and add corrections to the control signal. Another thing we have to take into account is the maximum task velocity of the robot, so when translated to joint velocity, it does not pass a certain limit. With the polynomial path planning the only way to reduce the maximum task velocity is to expand the time period of the path. But even so, the maximum velocity can be reached only once as depicted on the figures above. This is not efficient because most of the time, the robot does not move with maximum velocity. Owing to this an algorithm that will allow the robot to travel at maximum velocity most of the time is sought.

4.2.2 Potential Force Field

Another approach is to treat the robot's configuration as a point in a potential field that combines attraction to the goal. The resulting trajectory is output as the path. This potential field can also include repulsion from obstacles to achieve obstacle collision avoidance, but this feature is not in the context of this paper as it requires either the knowledge of the obstacle positions prior to the computation of the path or the use of a sensor that has a 360° window of observation for obstacle detection.

To use potential force fields, it is essential to have a method for localisation in order to calculate the force applied to the robot at the specific position each time step. So the algorithm is calculated inside the numerical integration function alongside with the calculation of odometry. The odometry is calculated based on the method described in 3.2.1.

At each time step:

$$R_{goal}=\sqrt{((x_c-x_{goal})^2+(y_c-y_{goal})^2)} \quad , \quad F_{att}=K_{att}\cdot R_{goal}$$
$$if\,(F_{att}>coef\cdot V_{max})\,then\;F_{att}=coef\cdot V_{max}$$
$$\theta_{goal}=atan2(y_{goal}-y_d\,,x_{goal}-x_d)$$
$$\Rightarrow F_x=F_{att}\cos(\theta_{goal}) \quad , \quad F_y=F_{att}\sin(\theta_{goal})$$
$$F_{\phi}=K_{att}(\phi_c-\phi_{goal})\Rightarrow if\,(F_{\phi}>coef\cdot\omega_{max})\,then\;F_{\phi}=coef\cdot\omega_{max}$$

and the calculation of the desired velocities is:

$$v_x[n+1]=v_x[n]+b(F_x-coef\cdot v_x[n])$$
$$v_y[n+1]=v_y[n]+b(F_y-coef\cdot v_y[n])$$
$$\omega[n+1]=\omega[n]+I(F_{\phi}-coef\cdot\omega[n])$$

where K_{att}, b, I and coef are constants defined to control the starting and stopping of the robot. We also notice that we can define the maximum linear and angular velocities.

Example with $x_0=20, y_0=-40, \phi_0=-\frac{\pi}{2}$, $x_{goal}=-30, y_{goal}=20, \phi_{goal}=\pi$, $K_{att}=1500,$ $coef=100, b=1, I=1$, $V_{max}=3\,, \omega_{max}=0.2$:

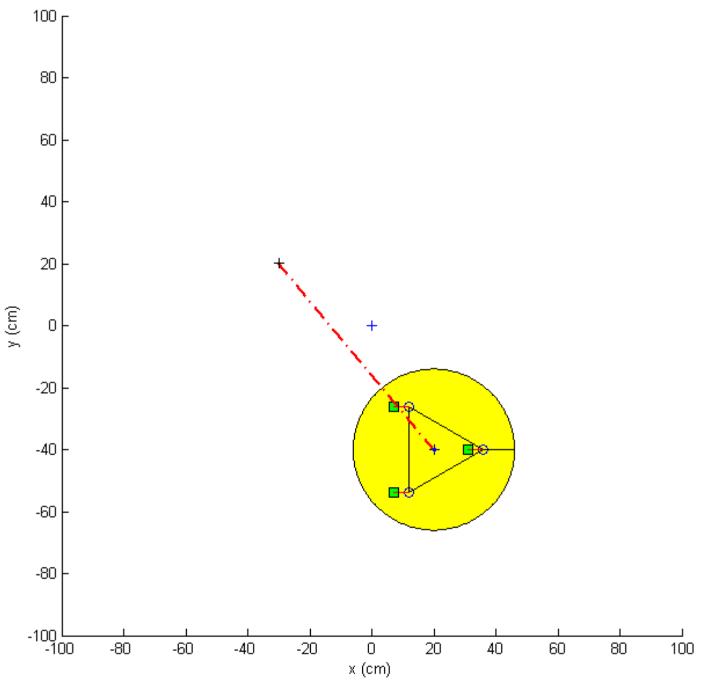


Fig 4.10 Robot's snapshot at t=0s

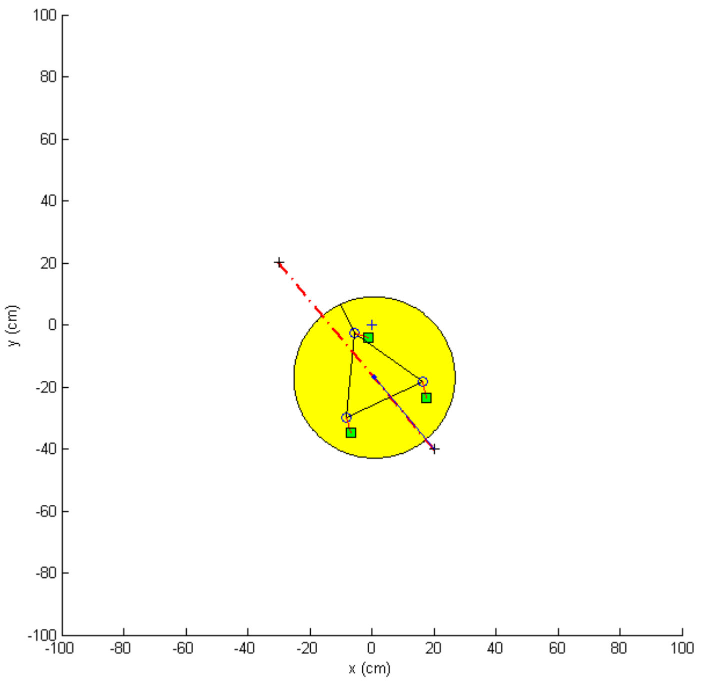


Fig 4.11 Robot's snapshot at t=10s

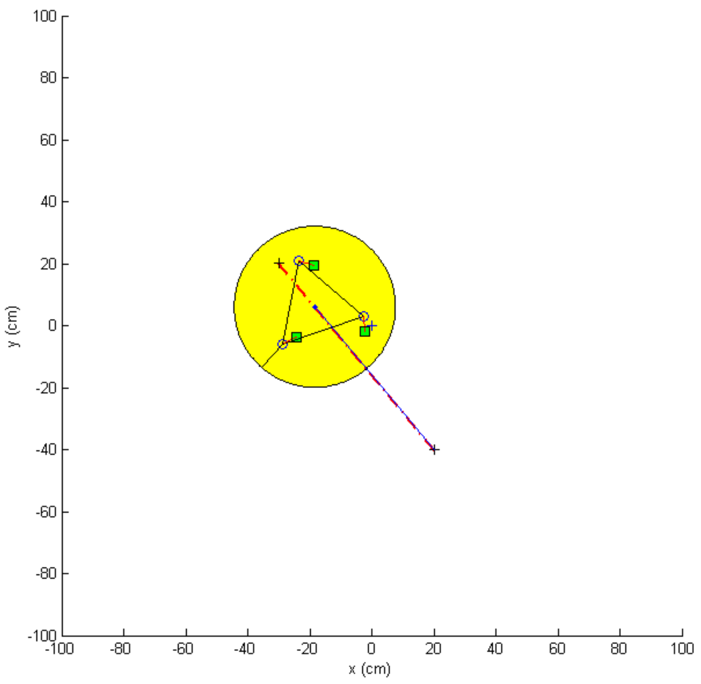


Fig 4.12 Robot's snapshot at t=20s

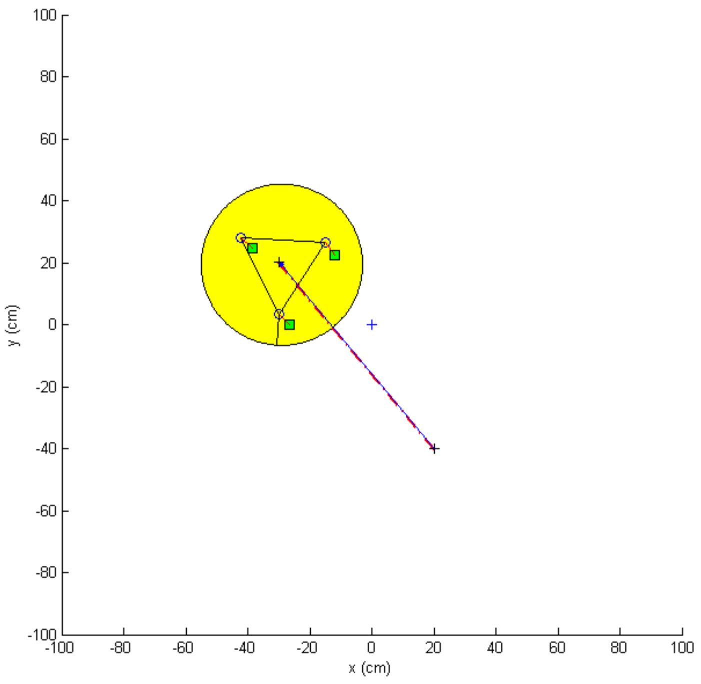


Fig 4.13 Robot's snapshot at t=30s

Like in the previous case we have the robot's snapshot each 10 seconds using the same symbols. The main difference here is that we do not have a path generation rather than just a goal position. The algorithm, though, forces the robot to move at the near-optimal path, which is the steepest-descent path in the artificial potential field and it is denoted by the red line. With other robots, like differential driving robots, the algorithm would force them to move in a circular path to turn the robot towards the goal and then move forward. Because of the omni-directional ability this is not necessary and the robot moves directly following the red line. Below is the robot's position in respect to time. The red line denotes the goal position.

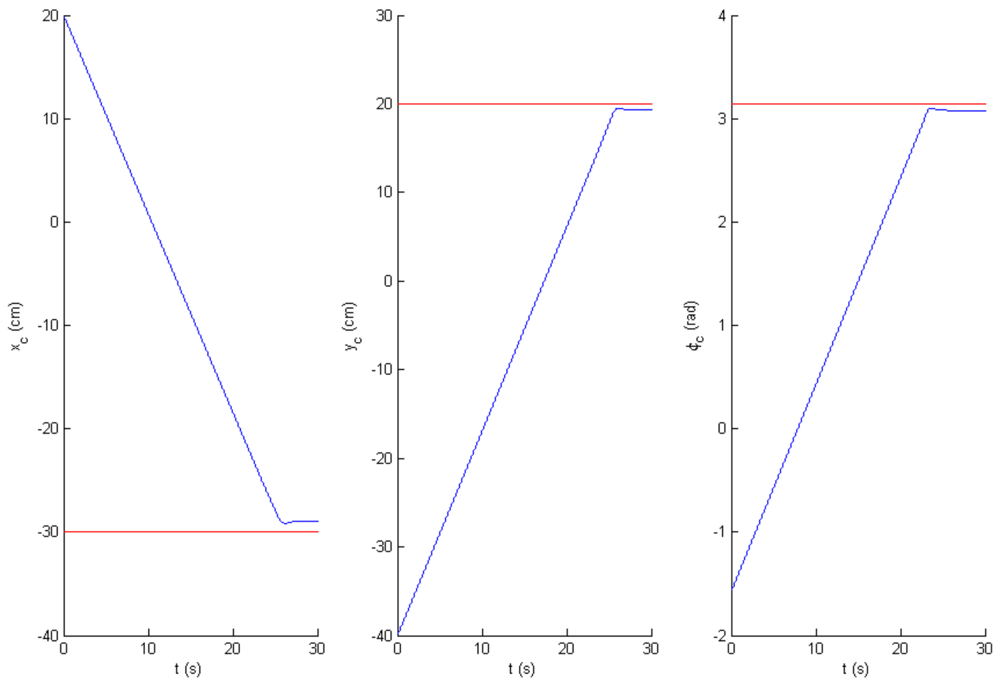


Fig 4.14 Robot's position

The robot follows the path with very few errors, due basically to odometry calculation. Another important observation is that the orientation of the robot reaches the desired orientation before the robot reaches the goal position. This might be not desired in specific movements.

The Joint Velocities produced by the Kinematic model are shown below.

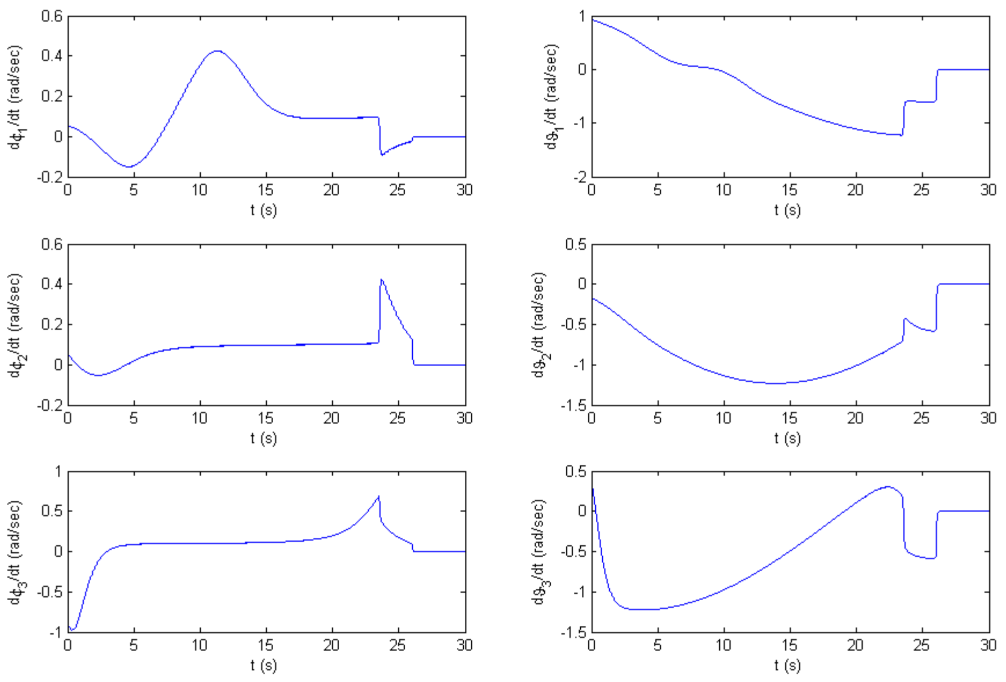


Fig 4.15 Joint Velocities

The robot's velocities calculated by the inverse K matrix and the joint velocities are shown below. Also there is a red line which denotes the desired maximum velocities which we set.

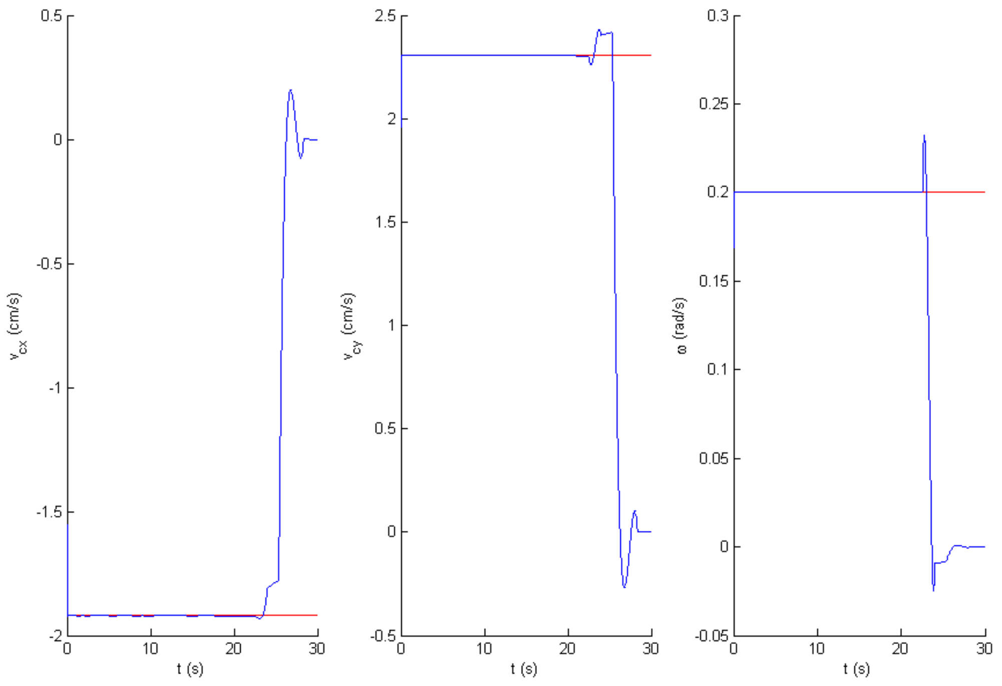


Fig 4.16 Robot's Velocities

The task velocities respect the maximum linear and angular velocities for most of the movement with a couple of spikes close to the goal. Specifically, the linear velocity of the robot, which is the vector sum of v_x and v_y, is equal to the maximum linear velocity. The problem that arise here is the very sharp acceleration in the start of the robot's movement and close to the goal position. The problem can be solved it by changing the K_{att}, b and I constants to achieve a smoother acceleration at the start and at the end of the robot's movement.

The Potential Force Field algorithm is an efficient algorithm for on-line path generation that produces an optimal path for the robot. But it lacks the ability to define multiple goal positions or to follow a defined path. Also, in the case of obstacle avoidance, the robot is very easy to get trapped in a local minimum.

4.2.3 Position-Velocity-Time (PVT)

The PVT algorithm converts a series of position-velocity-time pairs into motion frames that create the real-time command positions at each sample during the time intervals between the positions-velocity-time pairs. Actually, the algorithm fits a Jerk (non-constant acceleration) profile between user specified position-velocity-time points. For each point, the PVT algorithm calculates the Acceleration values to exactly hit the specified position and velocity at the next point. We divide time into events, and in each event we have a goal position-velocity-time that produces the desired acceleration:

$$\boldsymbol{A}[n]=\frac{\boldsymbol{V}(n+1)-\boldsymbol{V}(n)}{T[n]}$$

This acceleration is accumulated in the control signal each time step. After the event has passed (goal reached) we move on to the next goal. But the algorithm on its own is not able to achieve a trapezoidal velocity profile if the list of position-velocity-time is not properly computed to produce such a profile. This is done by interpolating several pseudo-positions between the initial position and the goal position and by calculating the time needed to travel between these positions related to the maximum velocity. So, we come up with a list of position-velocity-time pairs which, when used to compute the acceleration, produce a trapezoidal velocity profile with maximum velocity, the velocity we set.

$$\begin{cases} x_{path}=[linspace(x_{0,}x_d, intervals), x_d] \\ y_{path}=[linspace(y_{0,}y_d, intervals), y_d] \\ \phi_{path}=[linspace(\phi_{0,}\phi_d, intervals), \phi_d] \end{cases} \boldsymbol{p}[n]=\begin{bmatrix} x_{path}[n] \\ y_{path}[n] \\ \phi_{path}[n] \end{bmatrix}$$

$$\theta_{goal}=atan2(y_d-y_{0,}x_d, x_0) \Rightarrow \begin{matrix} V_x=V_{max}\cdot\cos(\theta_{goal}) \\ V_y=V_{max}\cdot\sin(\theta_{goal}) \end{matrix}$$

$$T[n]=max\{\frac{x_{path}[n+1]-x_{path}[n]}{V_x}, \frac{y_{path}[n+1]-y_{path}[n]}{V_y}, \frac{\phi_{path}[n+1]-\phi_{path}[n]}{\omega_{max}}\}$$

$$\boldsymbol{V}[n+1]=\frac{\boldsymbol{p}[n+1]-\boldsymbol{p}[n]}{T[n]}$$

We add an extra x_d, y_d, ϕ_d point in the end to set the final velocity to 0. The above algorithm has also the advantage that by increasing or decreasing the number of intervals the trapezoidal profile becomes sharper or smoother respectively, which affects the total time of the movement. Also, another advantage is that by adding points of a path into the x_{path}, y_{path}, ϕ_{path}, instead of linear spaced points, we generate a path following algorithm, not just point to point path. But, the main disadvantage is that the algorithm is an off-line path generator and it does not take into account the localisation data of the robot.

Nonetheless, we manage to find a way to integrate the localisation data into the algorithm to calculate corrections. First, we calculate the accelerations off-line, as described above. Then, at the end of each T[n], we compare the desired position of the current T[n] calculated by the algorithm with the current position calculated by odometry. If we find any discrepancies, we generate acceleration to minimise these errors, thus the algorithm becomes an online path generator with feedback from odometry.

At the end of T[n-1] and before the start of T[n] we calculate:

$$\boldsymbol{V}[n+1]=\frac{\boldsymbol{p}[n]-\boldsymbol{p}_{current}}{T[n]}\Rightarrow \boldsymbol{A}[n]=\frac{\boldsymbol{V}[n+1]-\boldsymbol{V}_{current}}{T[n]}$$

Example with $x_0=20, y_0=-40, \phi_0=-\frac{\pi}{2}$, $x_{goal}=-30, y_{goal}=20, \phi_{goal}=\pi$, $V_{max}=3$, $\omega_{max}=0.2$, $intervals=10$:

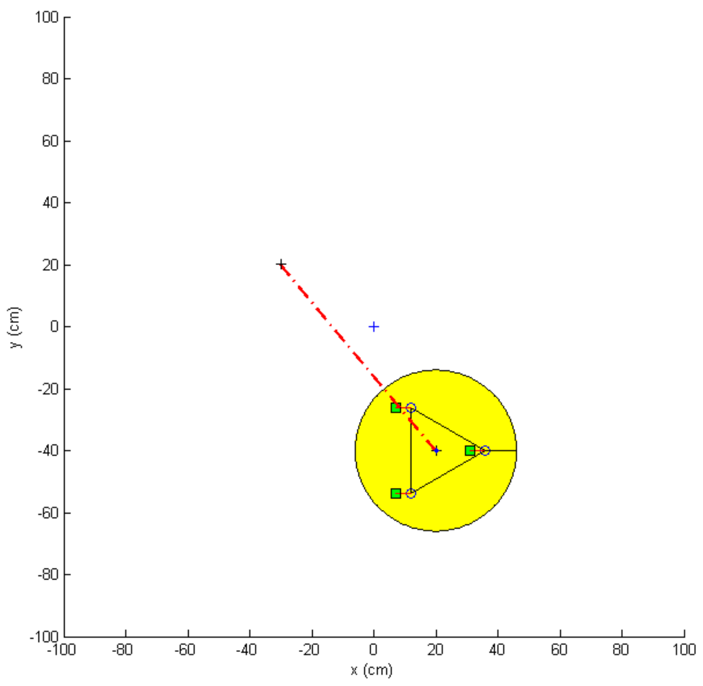


Fig 4.17 Robot's snapshot at t=0s

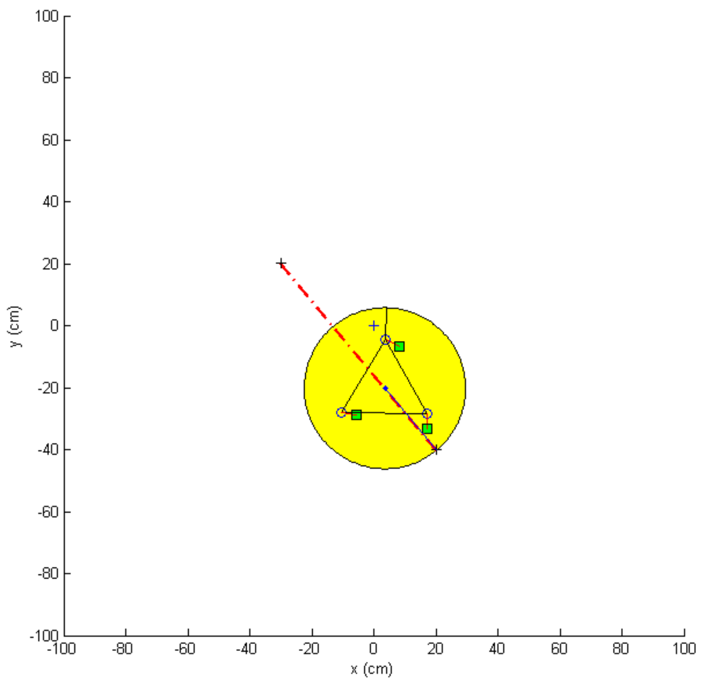


Fig 4.18 Robot's snapshot at t=10s

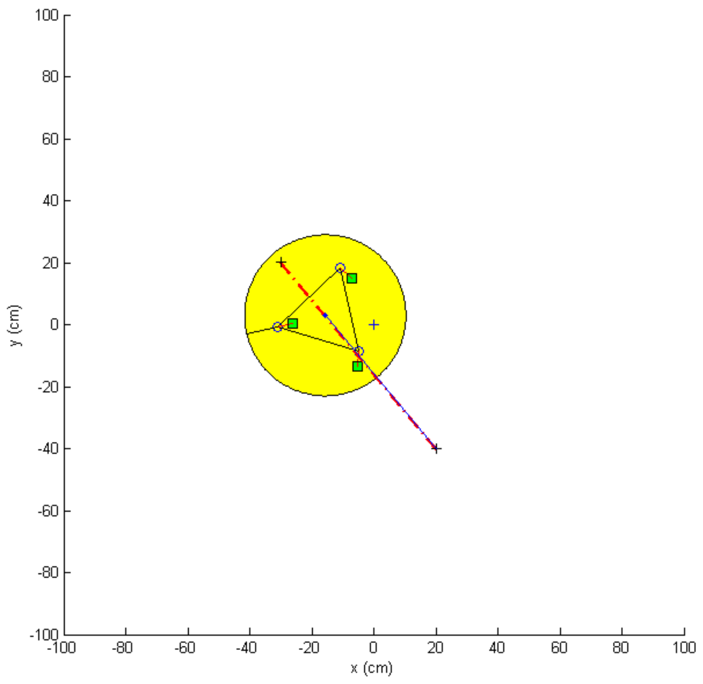


Fig 4.19 Robot's snapshot at t=20s

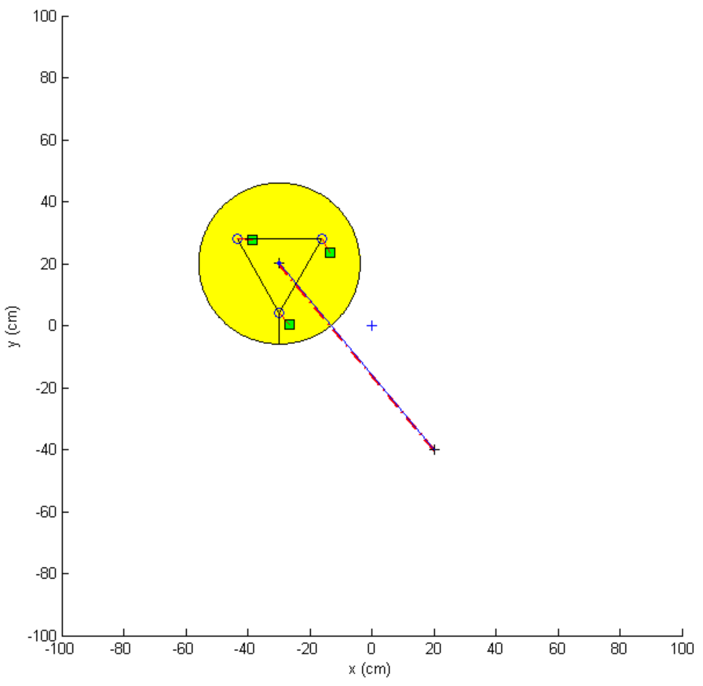


Fig 4.20 Robot's snapshot at t=30s

Following the same pattern as above we show the robot's snapshot each 10 seconds using the same symbols. Below are the figures of the robot's position in respect with time. The red line denotes the desired path calculated by the algorithm.

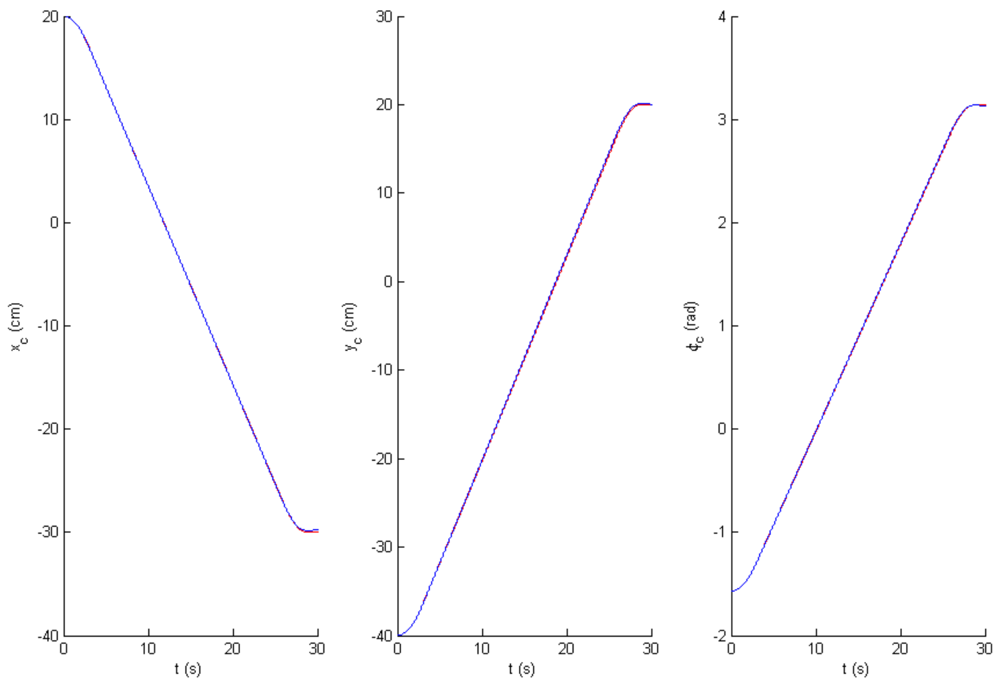


Fig 4.21 Robot's position

The Joint Velocities produced by the Kinematic model are shown below.

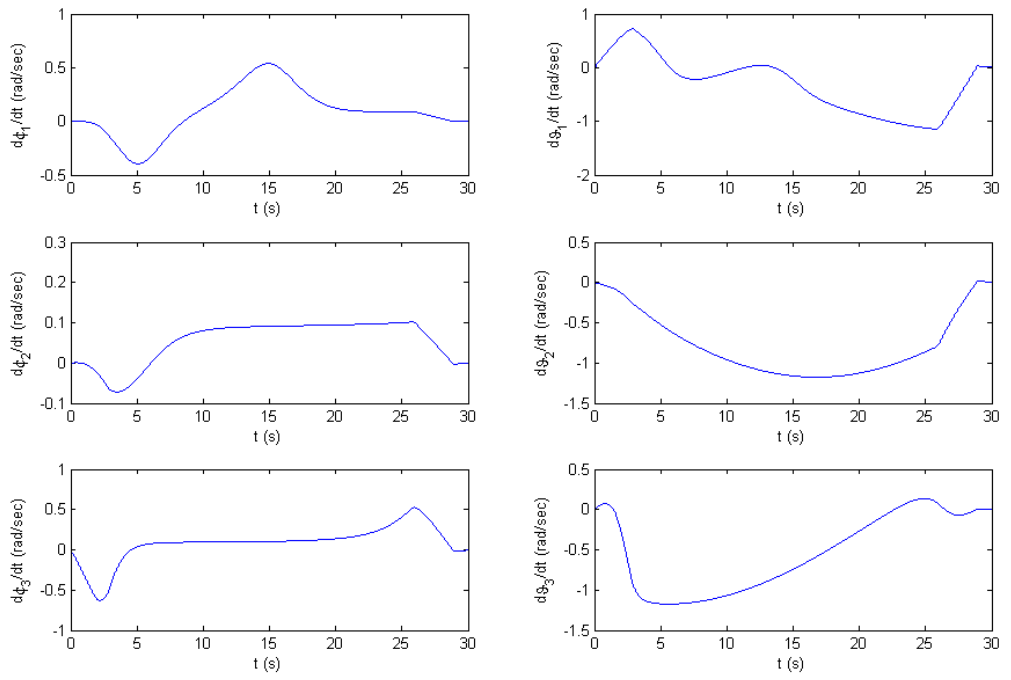


Fig 4.22 Joint Velocities

The robot's velocities calculated by the inverse K matrix and the joint velocities are shown below. Also there is a red line which denotes the desired velocities calculated by the algorithm.

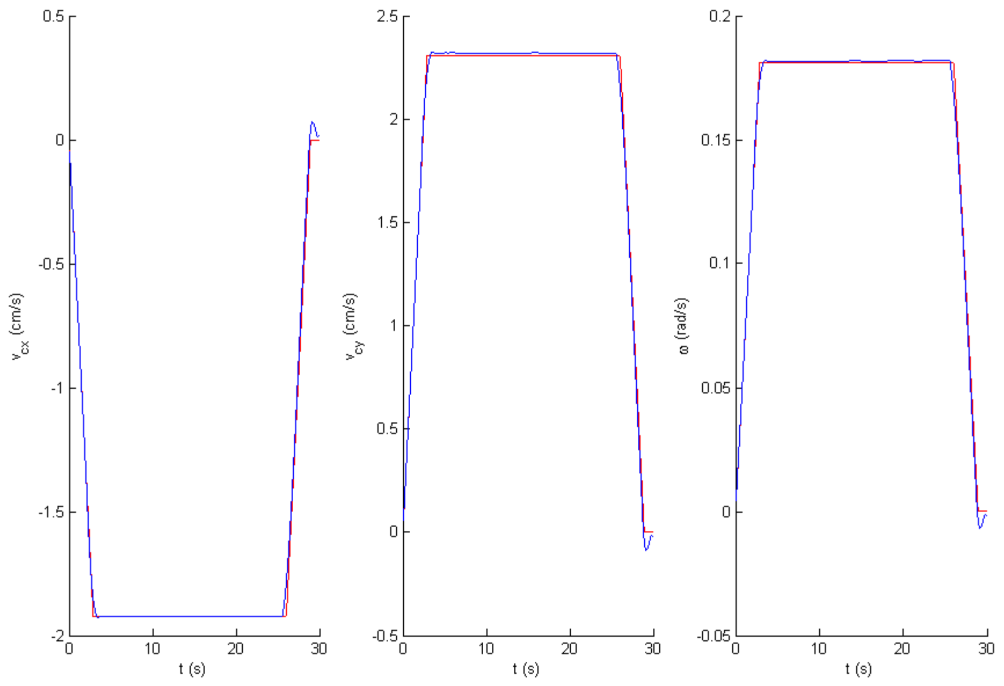


Fig 4.23 Robot's Velocities

The figures above illustrate that the robot complies better with this algorithm. The task velocities are smooth with a trapezoidal profile and the robot travels at maximum constant velocity for the largest part of the movement following the profile with a few errors. The angular velocity is not maximum in respect to the value we set because the time frame is bigger than the one needed for the maximum velocity travelling. This is due to the fact that we choose the maximum time frame per event of all points (x,y,ϕ). It is also noteworthy that the robot reaches the desired position at the same time it reaches the desired orientation.

Based on the above results and with the integration of the odometry data we have chosen to implement PVT algorithm on the robot. The lack of the need for obstacle avoidance has given us the opportunity to reject the Potential Force Field algorithm, while the Polynomial equation algorithm has been rejected for being an open-loop control algorithm.

Chapter 5
Hardware of the Robot

Each wheel of the robot consists of two stepper motors, two stepper drivers, an angle sensor, a micro-controller PIC and last but not least, the rubber wheel.

The θ_i variable, as described above, is actuated by the SOPROLEC 57HS09 stepper motor. The motor is wired in a bipolar parallel connection to the TECHLF SMD8A microstep driver. The ϕ_i variable is actuated by the TECHLF 23H056-8 stepper motor. Again, the motor is wired in a bipolar parallel connection to another microstep driver. Both drivers are then connected onto a micro-controller PIC that controls the speed and direction of each motor.

The cables which connect the devices are all mounted on the caster wheel, but the power cables have to come from the base of the robot. To ensure the full rotational ability of the wheel a slip ring is used.

5.1 Slip Ring

A slip ring is a method of making an electrical connection through a rotating assembly. It is a rotary coupling used to transfer electric current from a stationary unit to a rotating unit. This is accomplished by either holding the centre core stationary while the brushes and housing rotate around it, or holding the brushes and housing stationary while the centre core is allowed to rotate. Slip rings can be used where electrical power or signals need to be transferred to a rotating device, like in our case.

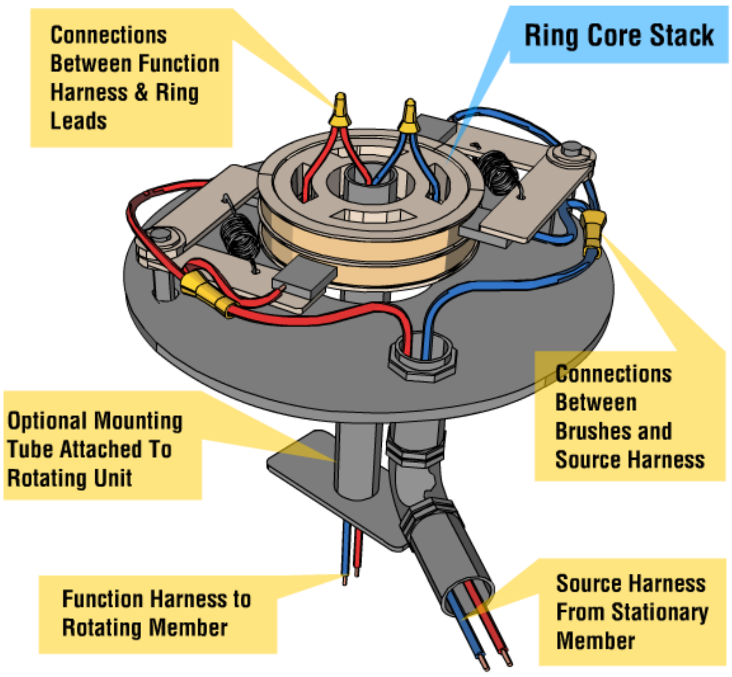

Connections Between Function Harness & Ring Leads
Ring Core Stack
Connections Between Brushes and Source Harness
Optional Mounting Tube Attached To Rotating Unit
Function Harness to Rotating Member
Source Harness From Stationary Member

Fig 5.1 Slip Ring example

In this work two different slip rings were developed. The first was a simple slip ring only to transfer the power to the wheels and powers the motors, the drivers and the micro-controller. Any other connectivity with the micro-controller has to be wirelless. It has only one cable, the VCC, passing through the slip ring. The GND is connected on the chassis of the robot. The second slip ring, with 5 cables passing, is capable of establishing USB connectivity with the micro-controller in addition to passing power.

5.2 Microstep driver

The Microstep driver TECHLF SMD8A is a driver capable of controlling a stepper motor and therefore it is the connection between the micro-controller and the motor. The driver is connected as shown in the figure below. The phases of the motor are connected directly to the driver. The micro-controller has 4 connections:

COM: The ground of the micro-controller
CLOCK: The control signal generated by the micro-controller
DIR: Direction signal
ENABLE: Enable signal

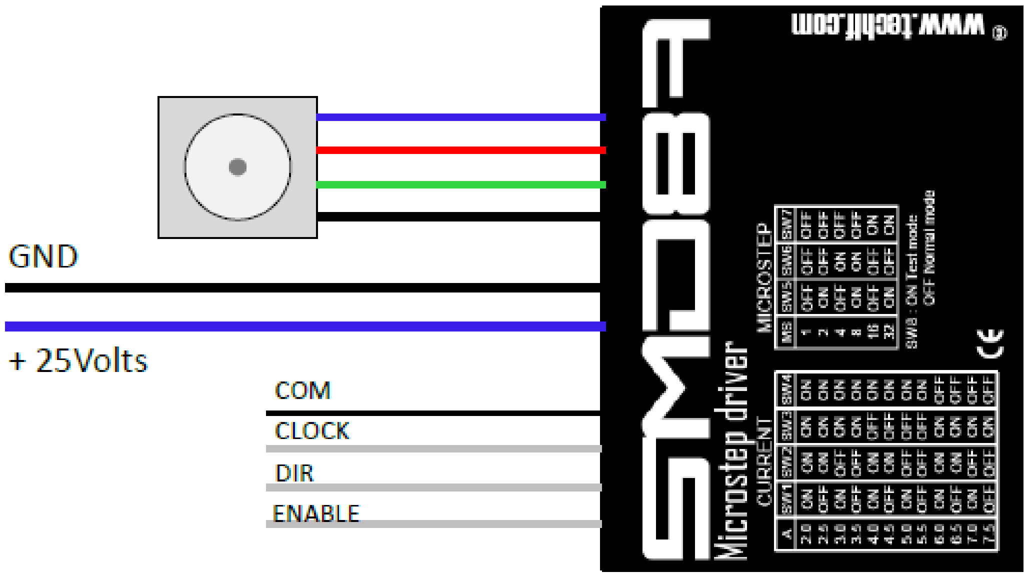


Fig 5.2 Microstep motor SMD8A pin diagram [Tech8A]

The driver sets the velocity of the motor according to the frequency of the CLOCK signal. The DIR signal sets the direction of the motor (1 one way, 0 the other way). The ENABLE signal is the on/off switch of the motor (0 on, 1 off).

5.3 Angle sensor

On the robot there are three angle sensors, one for each wheel. They are placed on top of the rotation axis of the wheel. The sensor consists of two parts. The first part is a small magnet that is placed directly on the rotation axis and it rotates with the axis. The second part is the austrianmicrosystems AS5045 chip. This is the actual sensor and is placed above the magnet mounted on the robot chassis. In this way the chip's rotation does not correspond to that of the robot.

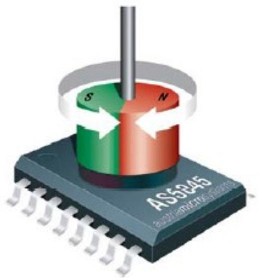


Fig 5.3 AS5045 Angle sensor [Aust45]

The sensor is a contactless magnetic rotary encoder for accurate angular measurement over a full turn of 360°. It is a system-on-chip, combining integrated Hall elements, analog front end and digital signal processing in a single device. The sensor measures the rotation of the magnet which is relative to the chip's centre as shown on the figure above. The absolute angle measurement provides instant indication of the magnet’s angular position with a resolution of 0.0879° = 4096 positions per revolution.

5.4 Micro-controller PIC

The micro-controller PIC that is mounted on each wheel is responsible for controlling the two motors. It is connected to both drivers and it outputs different control, direction and enable signals for each driver. The control signal (CLOCK) is a generated square wave signal with specific frequency related to the desired velocity.

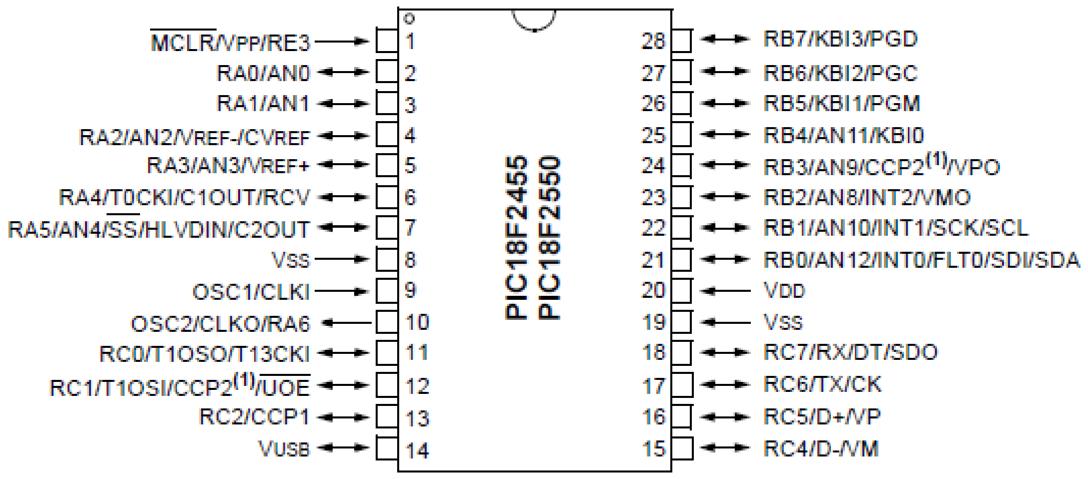


Fig 5.4 Mirco-controller PIC pin diagram [PIC18]

Owing to the use of a slip ring, the communication between the computer and the micro-controllers is either wireless or by USB depending on the slip ring we use. To add wireless communication ability to the micro-controllers we have used the ZigBee technology, which will be explained in a following paragraph.

5.5 ZigBee

To be able to have full rotary ability on the wheels using the simple slip ring but also have communication between the computer and the micro-controller, the communication must be wireless. So we have utilised the ZigBee technology. ZigBee is a specification for a suite of high level communication protocols using small, low-power digital radios for Low-Rate Wireless Personal Area Networks, such as wireless light switches with lamps, electrical meters with in-home-displays, and consumer electronics equipment via short-range radio needing low rates of data transfer. The technology defined by the ZigBee specification is intended to be simpler and less expensive than other WPANs, such as Bluetooth. ZigBee is targeted at radio-frequency (RF) applications that require a low data rate, long battery life, and secure networking. In our application we have used the Digi XBee chip shown in the figure below.

Fig 5.5 Digi XBee [XBee00]

The XBee chip connects to the serial interface of the micro-controller PIC. So the micro-controller “sees” only a serial communication with a device, so it does not need any special programming, other than the typical programming with serial communication. The micro-controller does not take part in the conversion of the wireless transmission to serial data. The XBee chip does not need any programming; in fact, it makes the conversion automatically. So the micro-controller receives the data and commands like serial in data. There is an XBee chip at each micro-controller. Also there is a pair of micro-controller and XBee chip that connects to the computer by USB. This micro-controller receives the data and commands from the PC through USB communication and sends them as serial out data to XBee which then transmits them to the other micro-controllers.

A problem that arises with XBee communication is that when the computer sends a command, every XBee receives it, so we cannot send commands or data to a specific micro-controller. A solution for this problem is described in paragraph 6.1.1. Also, an other limitation of XBee is that we can use it only to transmit or to receive data. We cannot use to for both. So we set the XBees of the wheels only to receive and the XBee of the computer only to transmit.

5.6 Caster Wheel Assembly

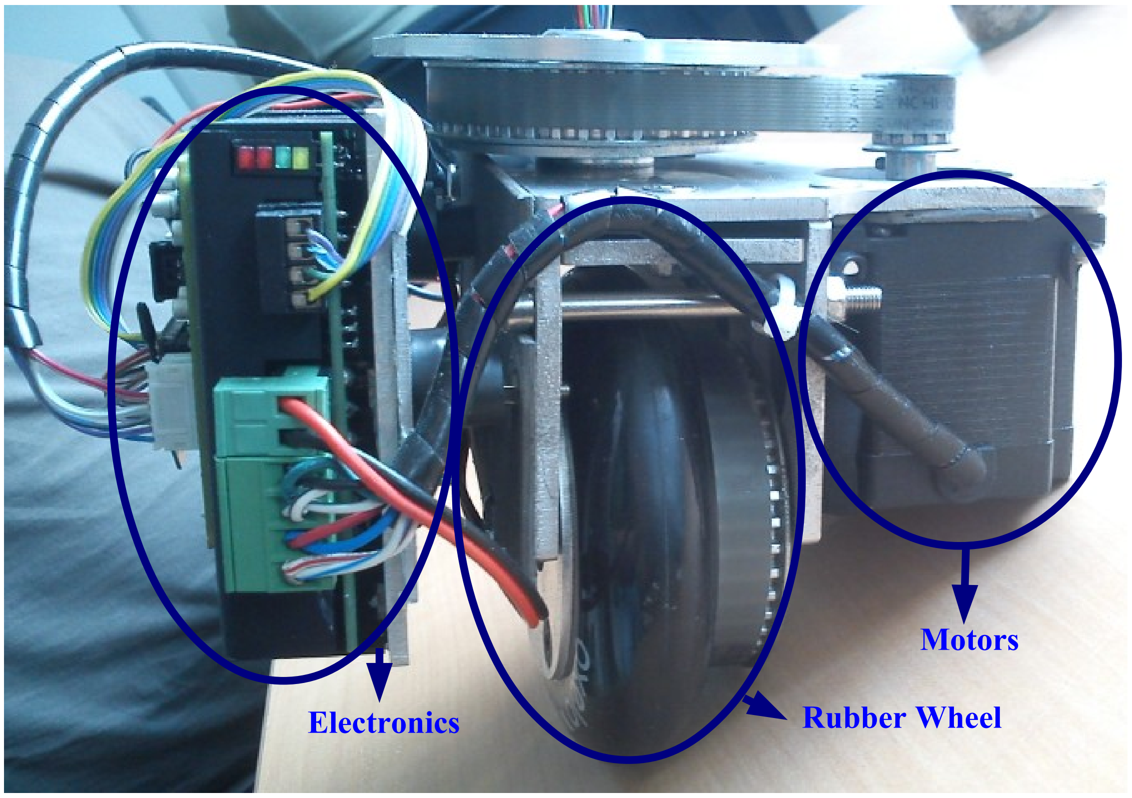


Fig 5.6 Caster Wheel Module

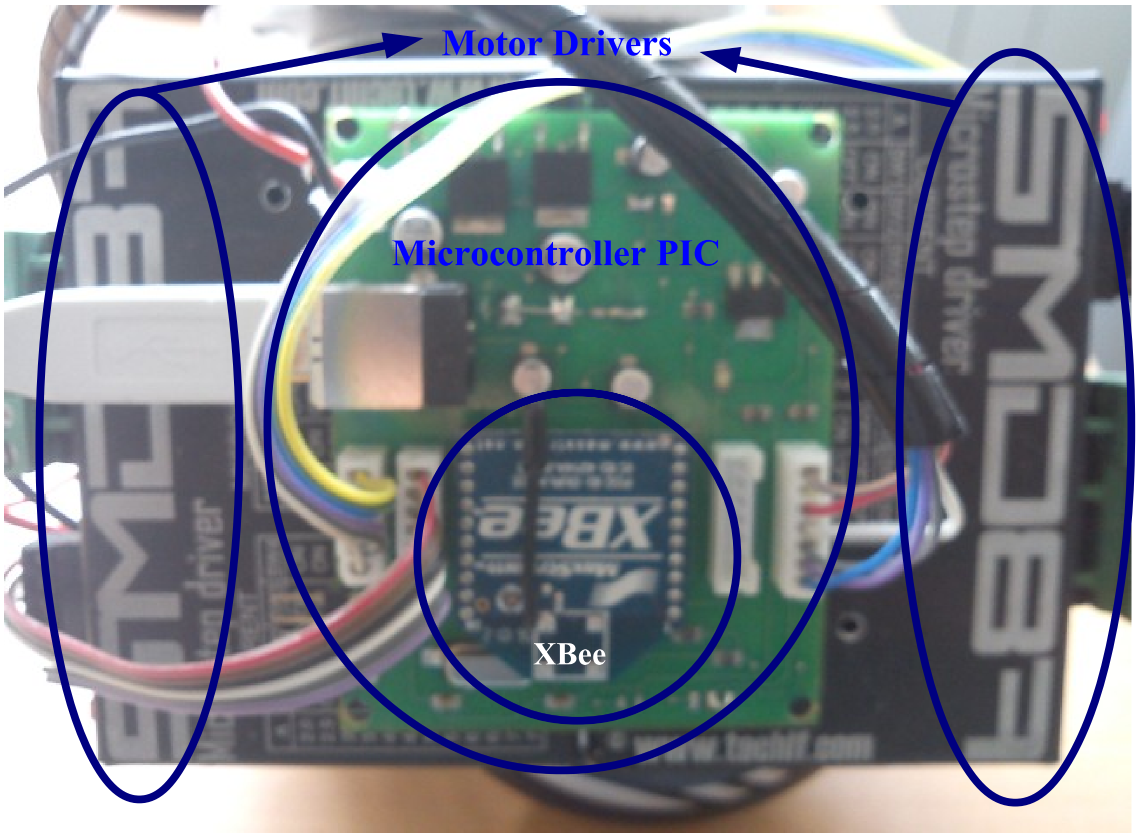


Fig 5.7 Electronics on the wheel

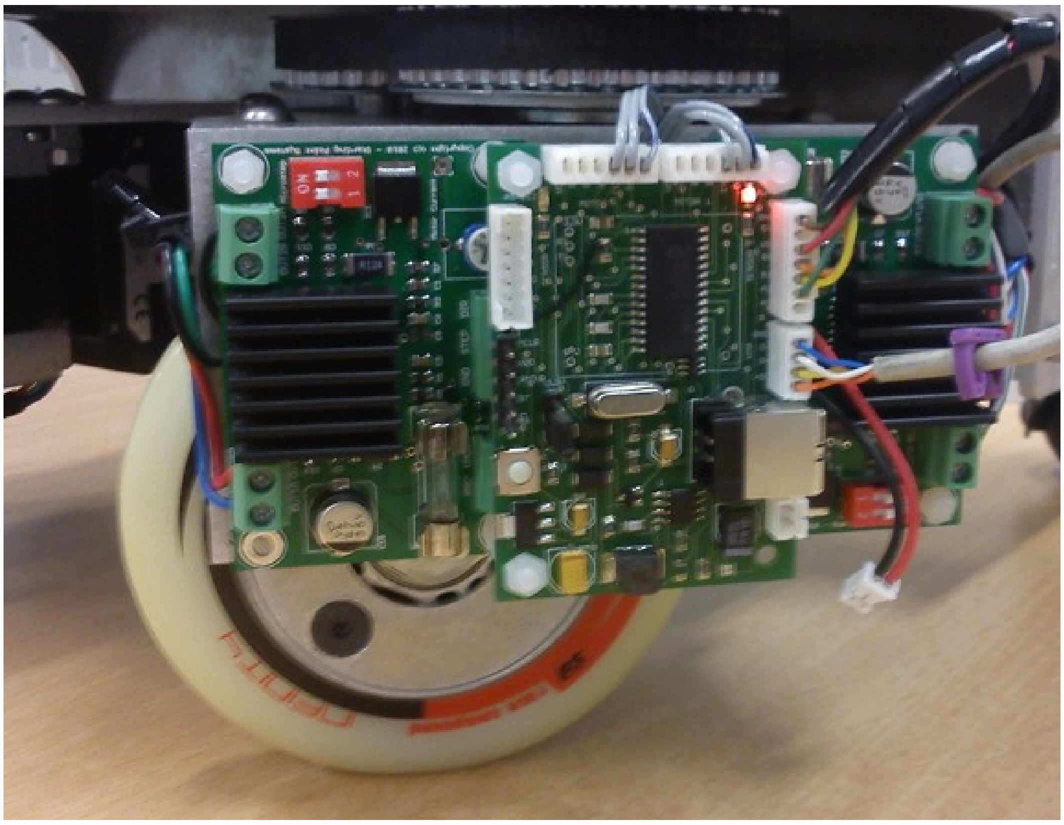


Fig 5.8 Other Caster Wheel Assembly

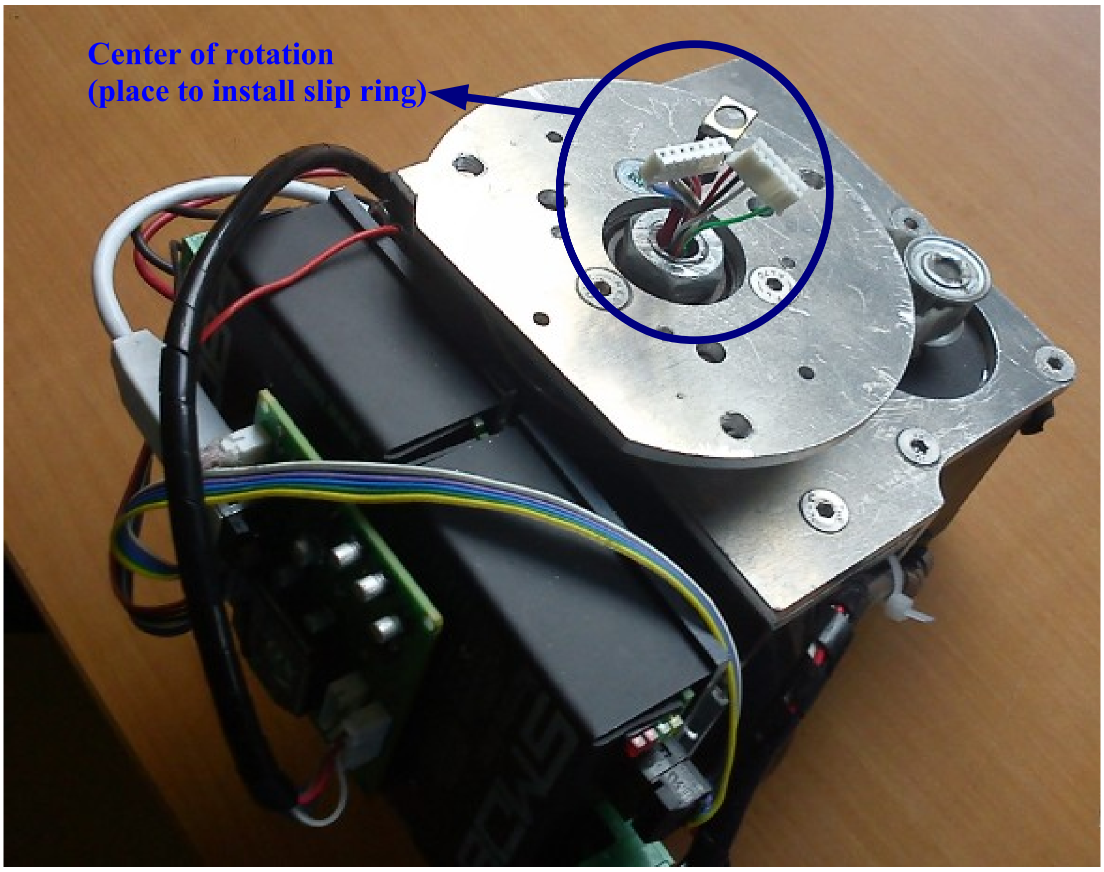


Fig 5.9 Top of caster wheel

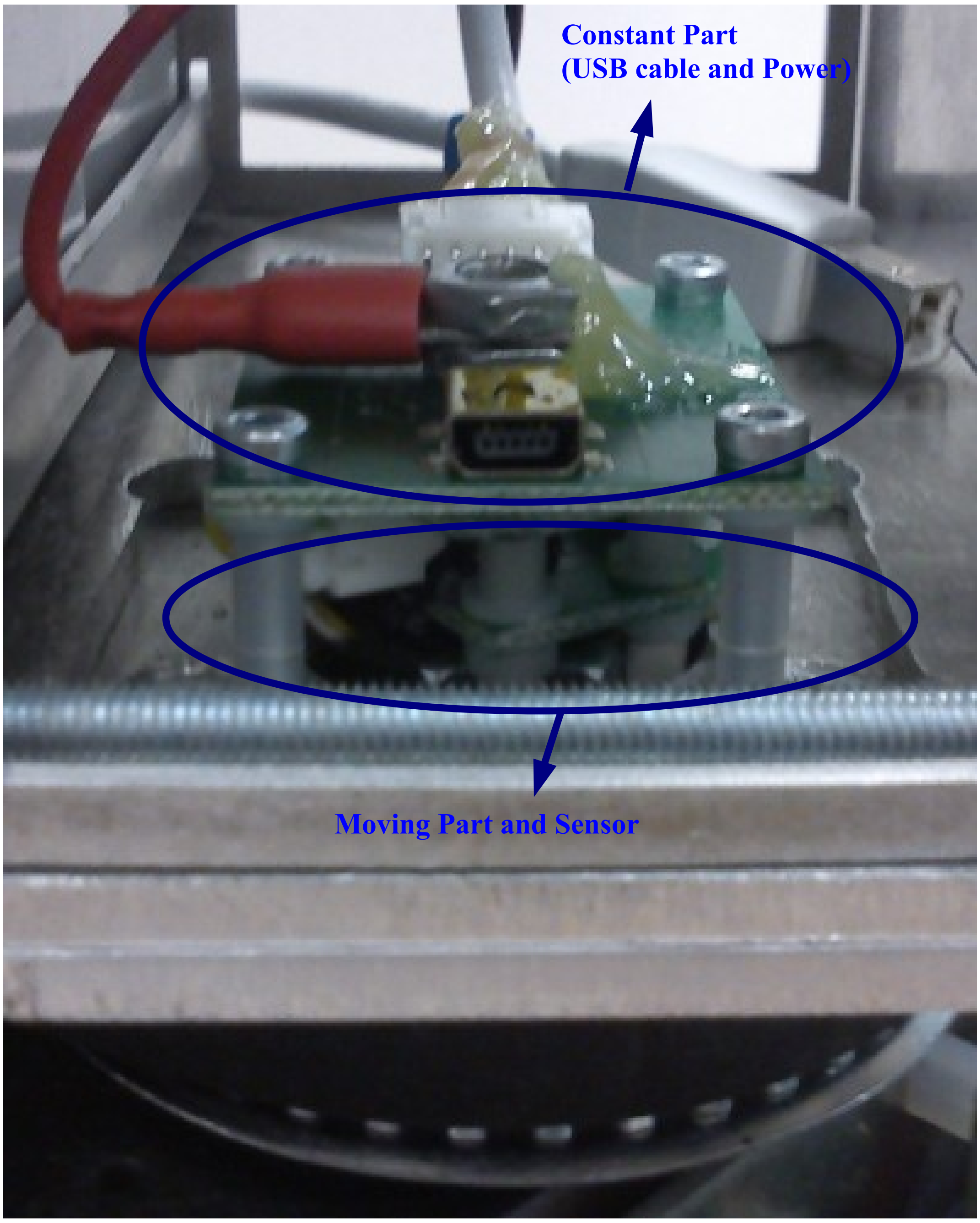


Fig 5.10 Slip Ring on robot

Chapter 6
Micro-controller Programming and User Interface

Based on the hardware described above, there are only two unique configurations. The first is with the use of the simple slip ring and the wireless communication between the computer and the micro-controllers. Due of the limitation of XBee only to transmit or receive data we have to use an other micro-controller to send the sensors readings to the computer by USB. This configuration is shown on the figure below and it is applied on the robot with the three wheels.

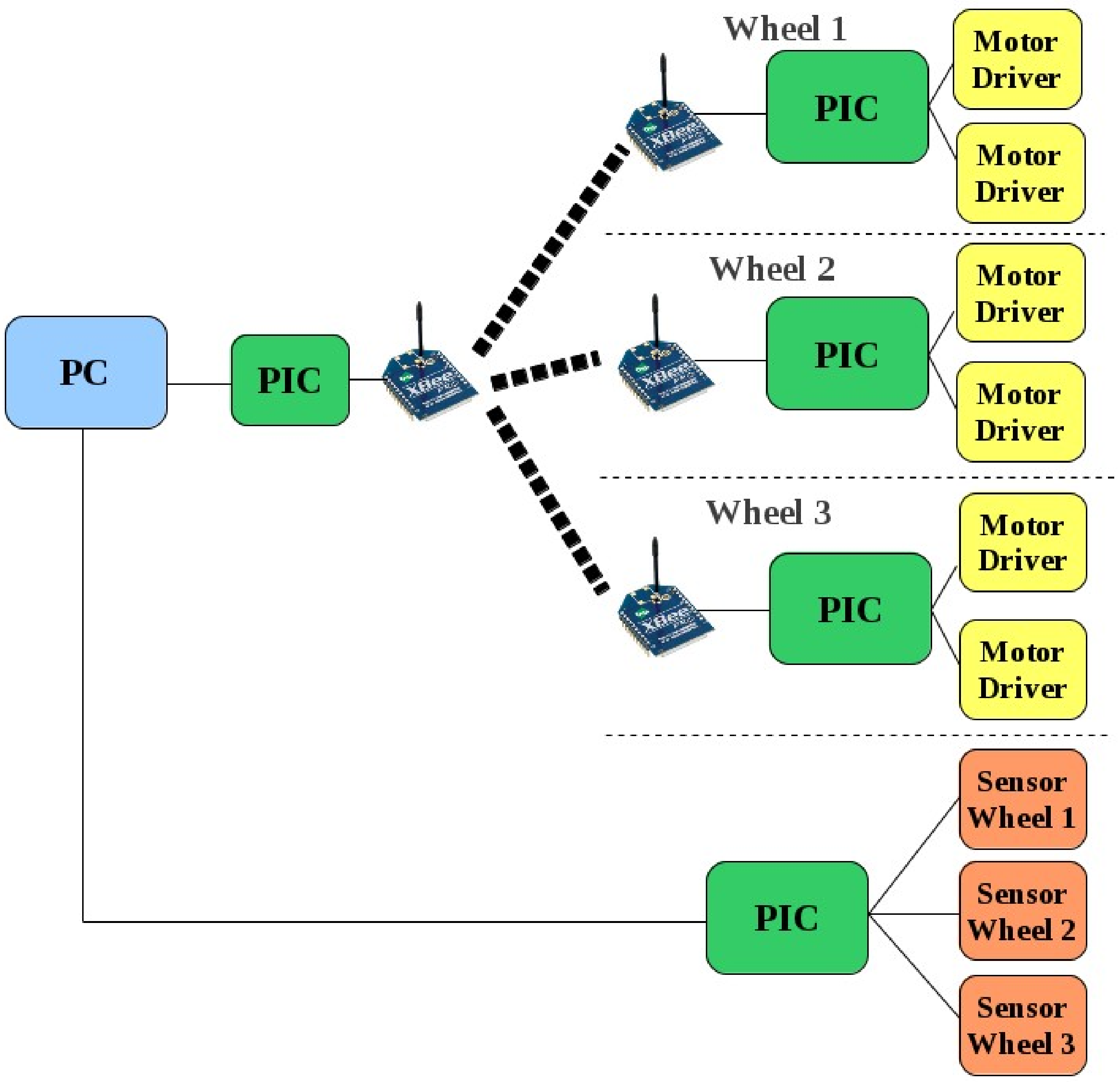


Fig 6.1 Connectivity configuration 1

The second configuration uses the second ring slip and all communication is done by USB. This provides us with the capability to send and receive data to the micro-controllers and so to connect the sensors at each micro-controller and eliminating the use of an extra micro-controller only for the sensors. This configuration is shown below and it is applied on the robot with the two wheels.

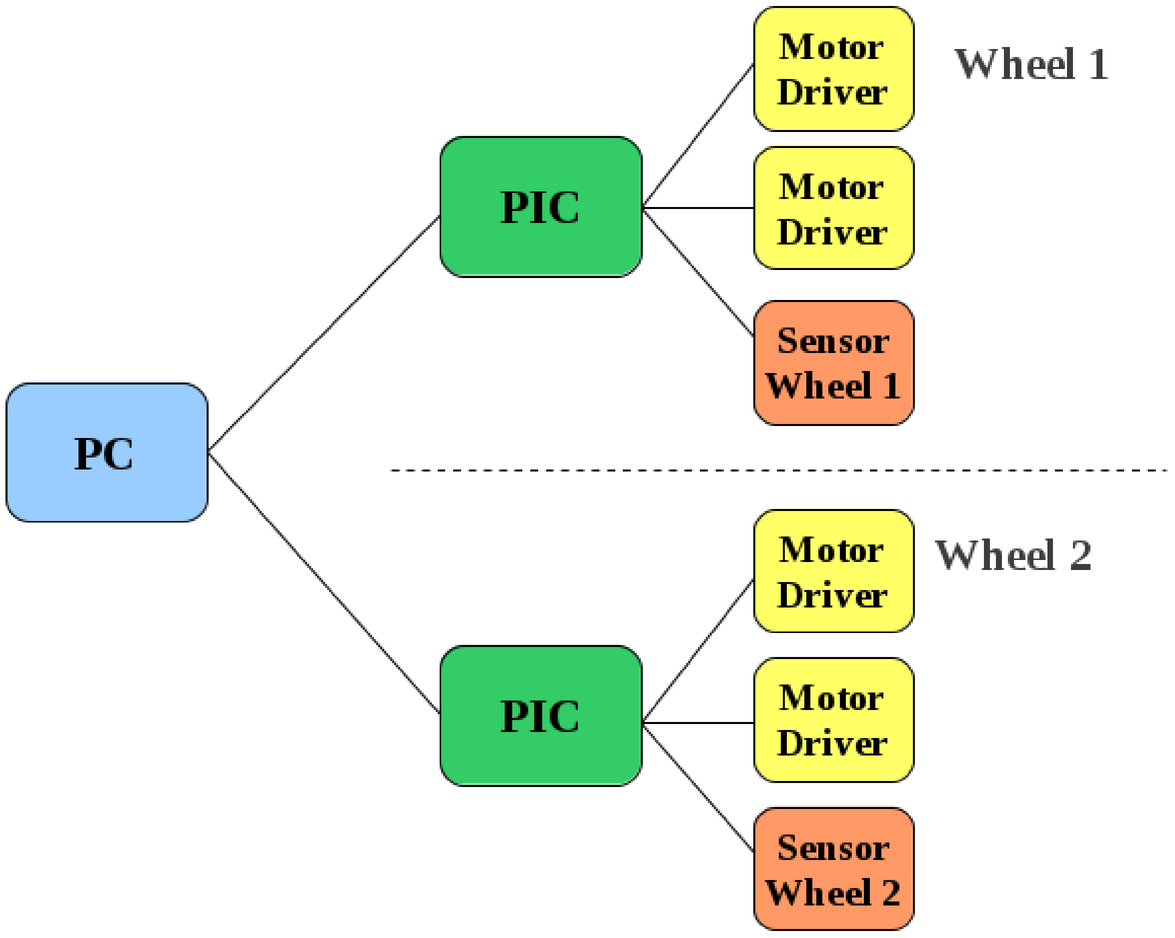


Fig 6.2 Connectivity configuration 2

The system is programmed to be able to function with both configurations, without the need for any modification.

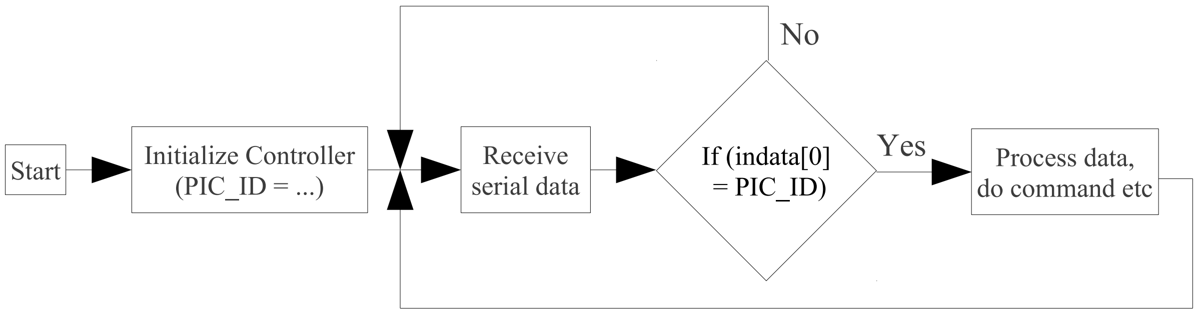
6.1 Programming of PIC Micro-controllers

To create a program for a micro-controller PIC we use the framework called MPLAB of the Microchip Technology Inc. The program is written either in C or Assembly. In our case, the programs were written in C. Next, we have used the CCS C compiler to compile the C programs to hex files, usable for the micro-controllers PIC. Finally, to program the micro-controller we have used the programmer PICkit3.

6.1.1 Control of Wheels

The basic functions that are necessary to be implemented are the serial/USB communication and the generation of two square waves of different frequencies. The data received by serial communication are 8-bit data and the data received by USB are 16-bit data. The generation of the waves is done with the CCP1/CCP2 pins of the micro-controller. The CCPx pin compares a set value with a micro-controller's timer and when equal, it toggles its state (0,1). This generates a square wave signal and the frequency can change by changing the value that is compared with the timer. To generate two square waves of different frequencies we just have to compare the CCP1 and CCP2 to different timers.

Besides the functions we wanted to achieve, the main goal was to have a single program for all micro-controllers of the wheels without having to change the code. But we also had to deal with the problem described above, that the ZigBee data is received by everyone. The solution which was proposed, and in the end implemented, was that each PIC has a unique ID number define as a constant in the program. In this way, the computer transmits, with the rest of data, the PIC ID of the micro-controller desired and the micro-controllers upon reception checks if the ID corresponds with their own and only the micro-controller that has the same ID continues to process the rest of the data.


Start
Initialize Controller (PIC_ID = ...)
Receive serial data
If (indata[0] = PIC_ID)
No
Yes
Process data, do command etc
Fig 6.3 Wheel's Micro-controller PIC program algorithm using XBee

For the USB communication the use of PIC_ID is not necessary since each USB device has its unique Product ID. So each PIC only receives its own control data. Furthermore, when we use USB connectivity we have also the sensor attached to the micro-controller and its reading are send to the computer. The algorithm for USB connectivity is shown below.

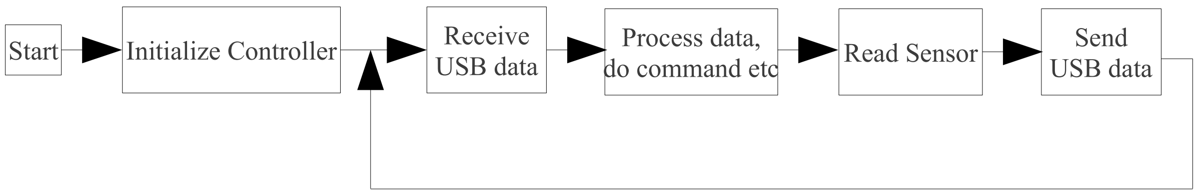


Fig 6.4 Wheel's Micro-controller PIC's program algorithm using USB

The two programs can be fused, as the processing of data is independent on how the data is received and the methods for receiving the data do not affect each other. The program is able to detect which method of communication is used. Several functions were developed, others for testing and others for proper functionality of the robot. The functions are characterised by a unique number which enables the computer program to choose which function to be executed by sending this number. The list of functions are SetMotor, SetMotorX and SetMotion. SetMotor is used to turn on/off both motors controlled by the micro-controller, SetMotorX sets on/off only a specific motor chosen by the computer and finally SetMotion sets the velocity and direction of a specific motor chosen by the computer.

The function SetMotion was designed to be used when the robot is in online mode (real movement), but a problem with the computer's program arose when we used wireless communication. The program could not send the commands for each micro-controller at the same timed event. The sending of the commands had to address all micro-controllers with the same transmission. So a new function was developed for this purpose. The function is global and has as a unique number a PIC_ID, and not a function number, to be able to be processed by all PICs with the same transmission. Specifically, a packet of 13 data (8bit) is transmitted as shown below:

```
    data[0] = OnlineState //PIC_ID = 4
    /* PIC_ID = 0 MOTOR = 0 */
    data[1] = Direction 1bit & Velocity 7bit (MSB)
    data[2] = Velocity 8bit (LSB)
    /* PIC_ID = 0 MOTOR = 1 */
    data[3] = Direction 1bit & Velocity 7bit (MSB)
    data[4] = Velocity 8bit (LSB)
    /* PIC_ID = 1 MOTOR = 0 */
    data[5] = Direction 1bit & Velocity 7bit (MSB)
    data[6] = Velocity 8bit (LSB)
    /* PIC_ID = 1 MOTOR = 1 */
    data[7] = Direction 1bit & Velocity 7bit (MSB)
    data[8] = Velocity 8bit (LSB)
    /* PIC_ID = 2 MOTOR = 0 */
    data[9] = Direction 1bit & Velocity 7bit (MSB)
    data[10] = Velocity 8bit (LSB)
    /* PIC_ID = 2 MOTOR = 1 */
    data[11] = Direction 1bit & Velocity 7bit (MSB)
    data[12] = Velocity 8bit (LSB)
```

Each PIC, after checking data[0] for OnlineState, reads the specific set of data based on its own PIC_ID and sets the control of each motor. As shown above,the possible velocities of each motor are 32767 = 15 bit number. In the case of USB communication, OnlineState function received the data and assign them accordingly without the use of PIC_ID.

6.1.2 Communication with a Host Computer

The programming of the micro-controllers that are connected to the computer is much simpler than the programming for the micro-controllers of the wheels. These micro-controllers are used only in the configuration with the wireless connectivity. It consists only of receiving or sending data from or to the computer respectively. A major deference is the USB connectivity which is realisable through specific libraries.

6.1.2.1 Sensors

The program for the micro-controller that reads all the readings of the sensors and sends them to the computer is very straightforward as seen in the figure below.

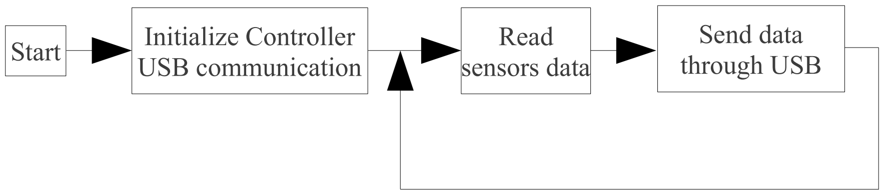


Fig 6.5 Computer's Micro-controller PIC program algorithm for sensor reading

6.1.2.2 Transmitter

The same applies to this micro-controller. At each time step the micro-controller receives data from the computer from the USB port and sends them to the serial port. Then the data are received by the XBee and automatically transmitted to all other micro-controllers.

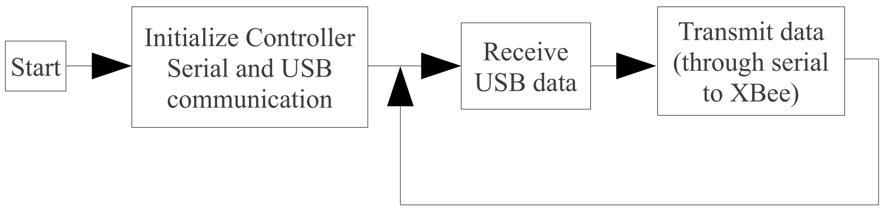
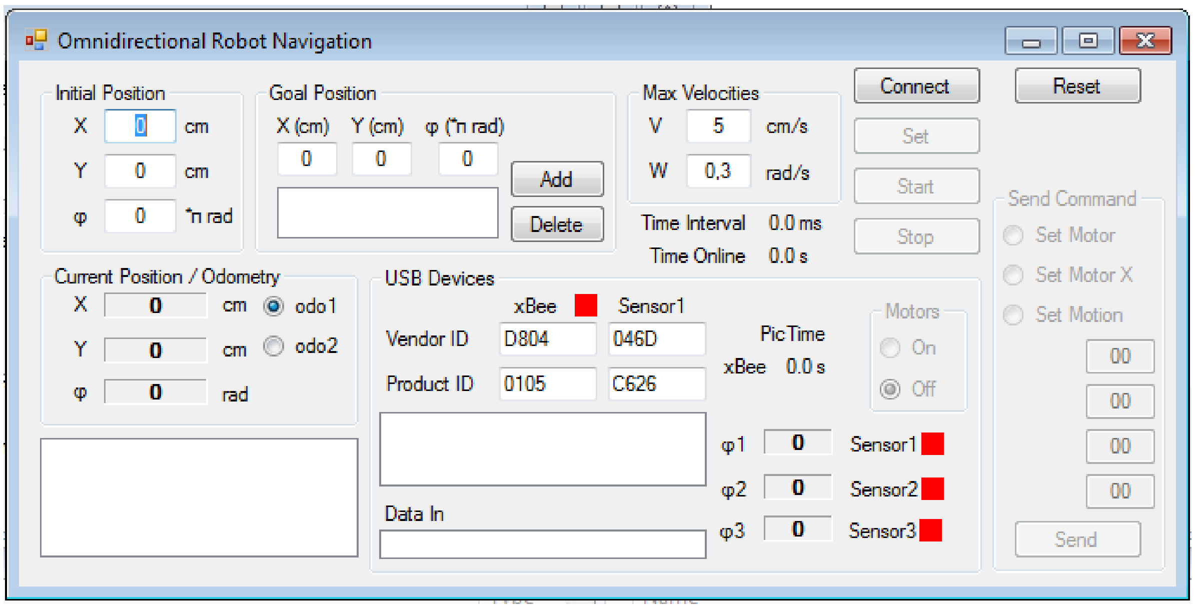

Start
Initialize Controller Serial and USB communication
Receive USB data
Transmit data (through serial to XBee)

Fig 6.6 Computer's Micro-controller PIC program algorithm for XBee transmission

6.2 User Interface

The biggest part of the programming section was the development of a User Interface program that would give the user the ability to define goals, send commands to the controllers and observe the movement of the robot. In addition, the program had to compute all the equations, like in the simulation, which include the calculation of the trajectory planning algorithm and the extraction of the desired velocities, the calculation of the odometry data, the calculation of the inverse Jacobian matrix and the extraction of the control signal to be send to the micro-controllers. Finally, the program had to communicate with the two micro-controllers attached to it, the transmitter and the sensor reader.

The programming language that was chosen to develop the User Interface is C# because it includes a ready USB library, which enables the program to communicate with the USB devices. Furthermore, C# is an object-oriented language that makes it very easy to create windowed programs with graphical interface. A sample of the program is shown below:



Initial Position: X 0 cm; Y 0 cm; φ 0 ^π rad

Goal Position: X (cm) 0; Y (cm) 0; φ (^π rad) 0

Max Velocities: V 5 cm/s; W 0,3 rad/s

Time Interval 0.0 ms
Time Online 0.0 s

Current Position / Odometry: X 0 cm; Y 0 cm; φ 0 rad; (● odo1) ○ odo2

USB Devices: xBee 🟥 Sensor1

	xBee	Sensor1
Vendor ID	D804	046D
Product ID	0105	C626

PicTime: xBee 0.0 s

φ1 0 Sensor1 🟥
φ2 0 Sensor2 🟥
φ3 0 Sensor3 🟥

Motors: ○ On (● Off)

00
00
00
00

Fig 6.7 User Interface program

6.2.1 Initial & Current Position – Odometry

In these sections, the user can define the initial position of the robot and then, when the robot is moving, observe its current position, which is calculated by odometry. At each time step, odometry is calculated as described in paragraph 3.2. We calculate the robot's centre position relatively to each wheel and then take the mean value. The orientation of the wheel is calculated by adding the angular velocity each time step, which is calculated based on the joint velocities and the sensor readings.

6.2.2 Goal Position – Max Velocities

Here, the user can add or delete sets of goal positions (x,y,ϕ) in a list. Also, he/she sets the maximum linear and angular velocities. When the Set button is clicked, the program automatically computes the PVT algorithm, explained in paragraph 4.2.3, based on the first goal position in the list and the maximum velocities set by the user. After the robot starts the movement and when it reaches the goal position, the program checks the list for a next goal position so as to either compute the PVT again or stop the movement. When the program computes the PVT algorithm, it actually builds a path with a specific number of intervals between the current position and goal position, calculating the T, the velocities and the acceleration for each time event (see 4.2.3).

6.2.3 USB Devices – Send Command – Motors

The USB Devices sections contains all the information required about the connectivity with the USB devices. It indicates the existence of a device along side with the PIC's online time and the sensors readings. The Send Command and Motors sections are activated only when the xBee PIC micro-controller is present and they are used to send individual commands or to turn on/off all of the motors. The Vendor and Product ID are numbers defined in the USB libraries used for the connectivity.

6.2.4 Buttons

The buttons, which exist in the program, are Connect, Set, Start, Stop and Reset. The Connect button calls the function to connect the program with the USB devices. The Set button sets the initial positions, initialises the odometry data, checks if there is at least one goal position and call the function to compute the PVT algorithm. The Start and Stop buttons starts or stops the movement of the robot respectively and they are enabled only when connection to the USB devices is successful and the Set button is clicked. Finally, the Reset button resets all configurations and clears the goal list. Therefore, after clicking the Reset button the user must re-enter the initial positions, add goals and click the Set button before restarting the movement of the robot.

6.2.5 OnTimedEvent

This is the main function of the program which is called on every time step. In the function, the program updates the odometry data, reads the sensors, checks the PVT data and calculates the control signal. Then it sends it to the xBee transmitter. In addition, it updates several output texts to notify the user of the movement of the robot.

Chapter 7
Experiments

7.1 Versatility

The first cycle of experiments was to test the versatility of the system and the ability to control different robot configurations. At first we applied the system on a robot with three caster wheels. The configuration of the robot was as shown in Figure 6.1. So in total we had to control six motors, three translational and three angular motors. At the time of the experiment, the robot did not had sensors to measure the angles of each wheel. To overcome this, we set the initial angle of each wheel and then we integrated the angular velocity of each wheel to compute and estimate the angle of each wheel at each time step. The result was to have many errors in the direction the robot was moving relatively to the desired direction.

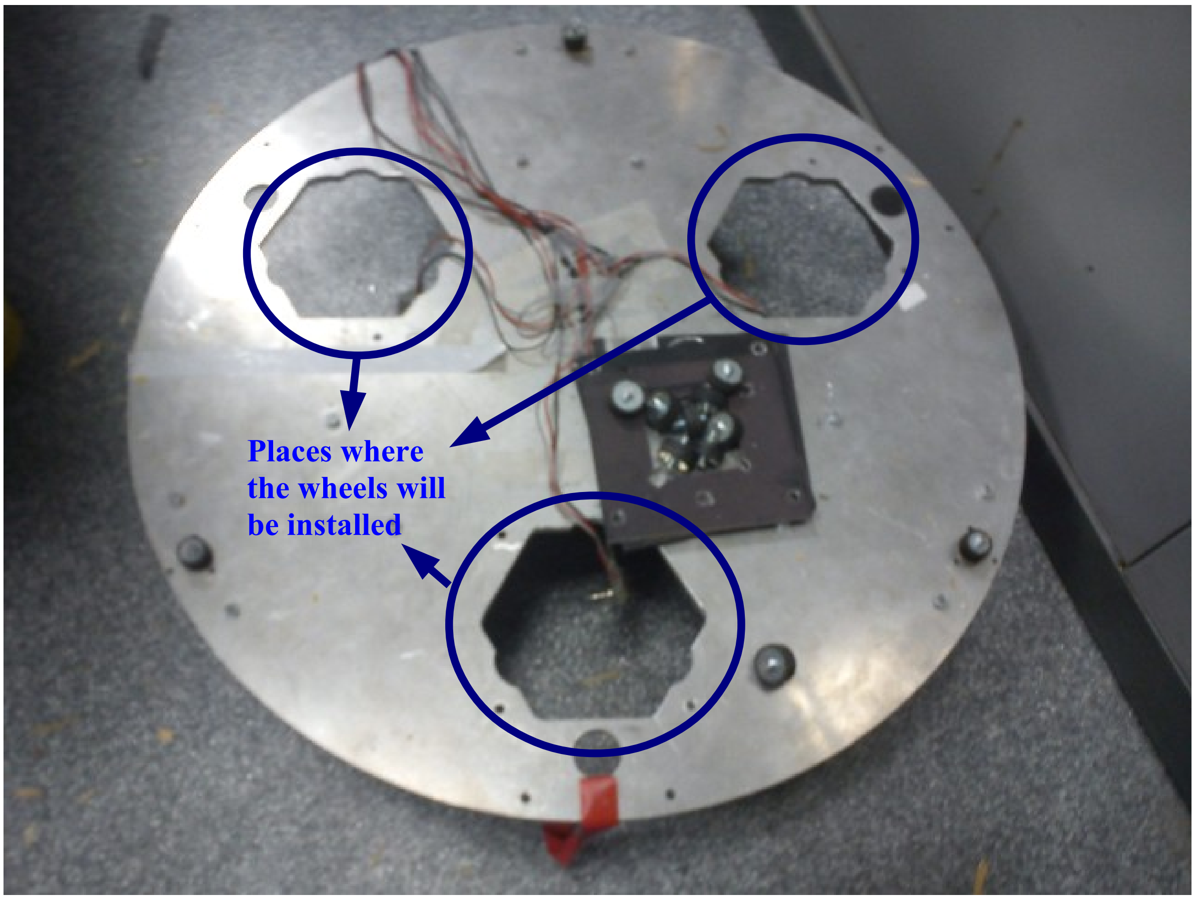


Fig 7.1 1st Robot's Chassis

Despite the errors, the system was found sufficient to control the robot, to start/stop the motors, turn the wheels in the correct direction and to communicate through Xbee. The trajectory planning algorithm and the odometry functioned properly, however, their results had errors that was produced by the lack of angular sensors.

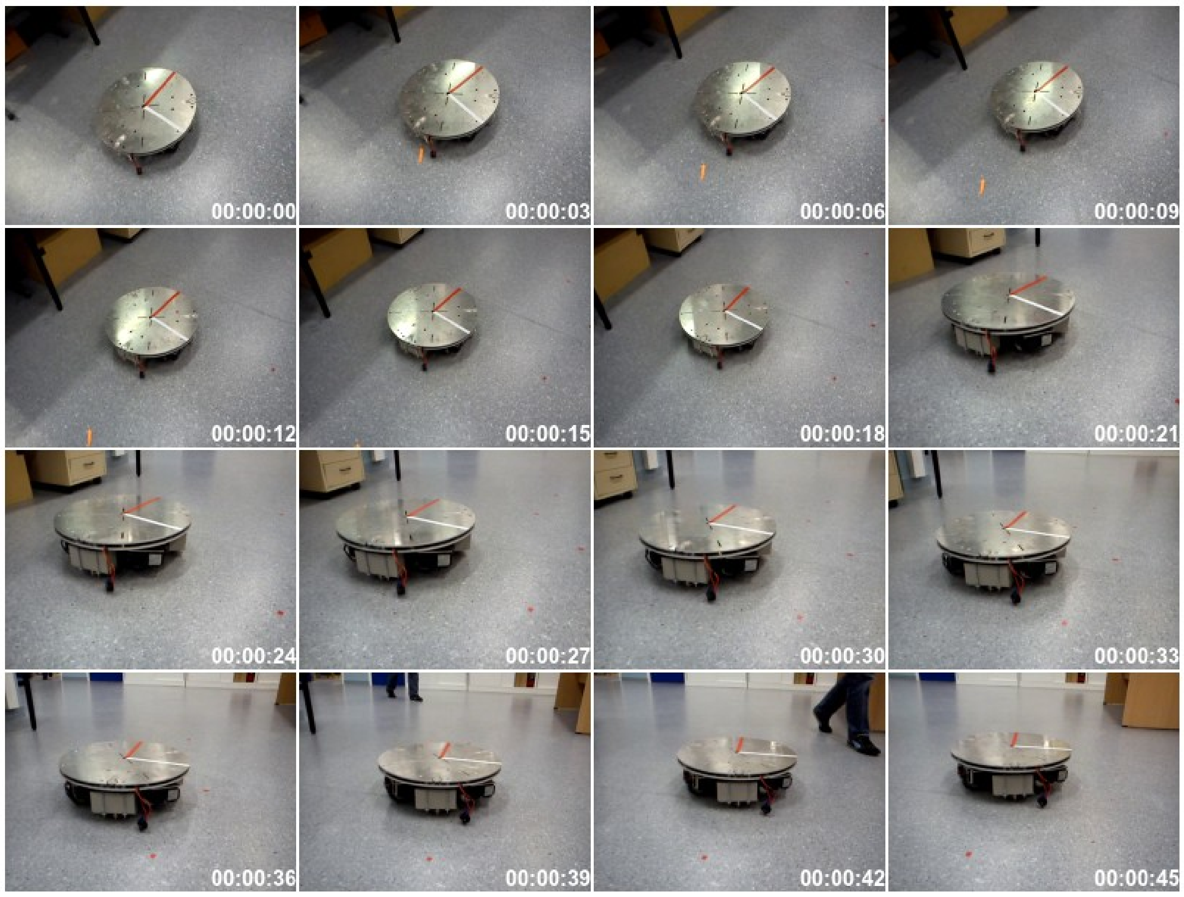


Fig 7.2 1st Robot's Movement

The second robot that was used to test the system, was a robot with two caster wheels. The configuration of the robot was as shown in Figure 6.2. The robot was equipped with angle sensors for both wheels and slip ring that permitted USB communication. So the use of Xbee was unnecessary. The system did not need to be modified extensively. The only two things that had to be altered were the kinematic model used for odometry and the computation of the joint velocities. We had to subtract the lines of the third wheel and change the dimensions of the robot. Finally, we chose the function to send the commands by USB. The system controlled the robot the same as before, by starting and stopping the motors and turning the wheels in the correct direction. The communication by USB and the reading of the sensors were successful. The movement of the robot was more accurate than before, but errors were again obvious.

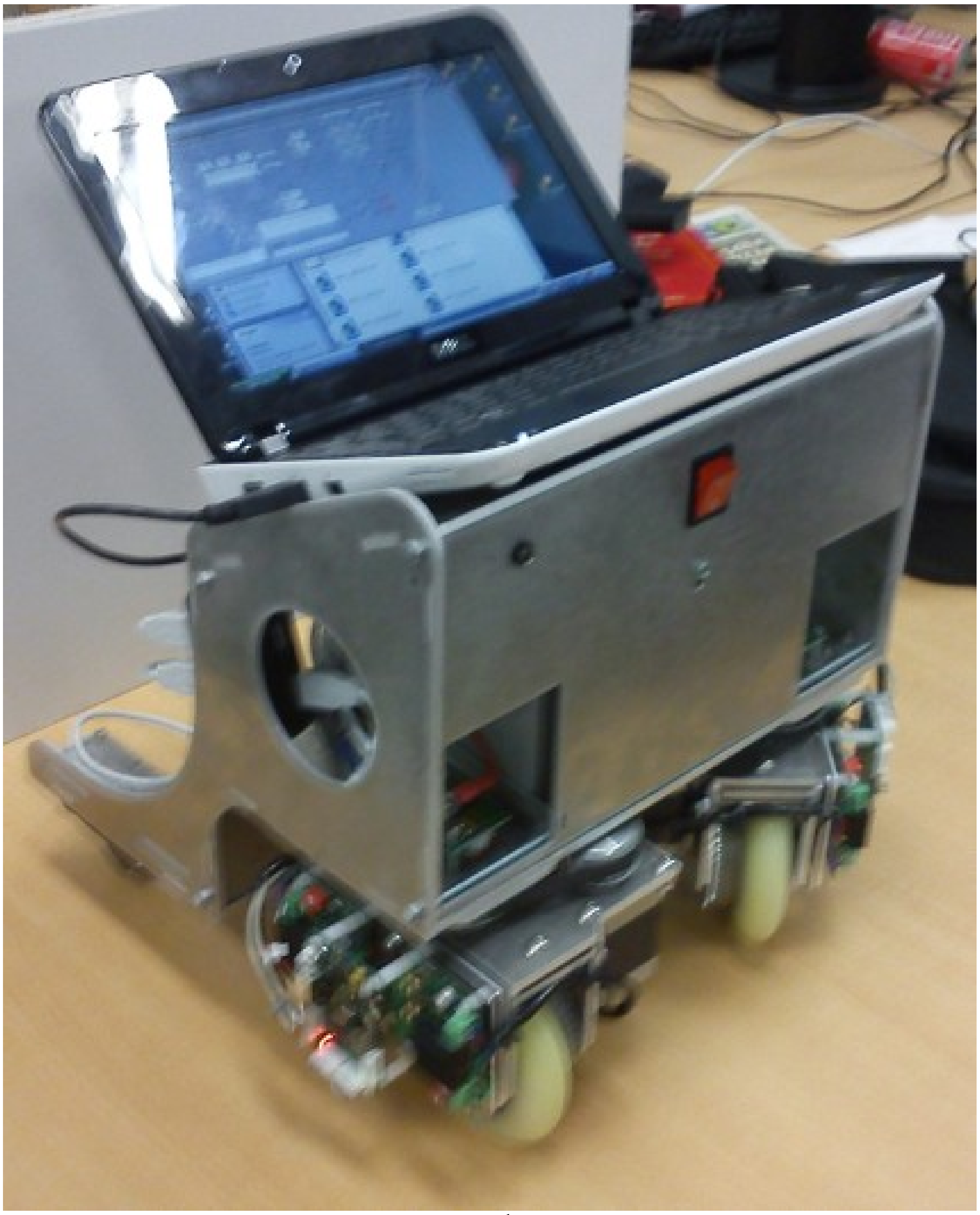


Fig 7.3 2nd Robot

This set of experiments gave us the opportunity to study the versatility of the system to control robots with different configurations. This proves that the block programming of the system works and it gives us the ability to use it on different robots. But also it revealed to us difficulties we could come across and limitations we have to take into account. The most important limitation was the time interval between the calculation of odometry and new control signal to be send to the motors. The connectivity, USB or Xbee, introduced a delay of 30-40 milliseconds between calculations. If we set a smaller interval the communication blocked and the system would stop to function. The interval plays a big part also in the calculation of odometry and it produces errors in the movement of the robot. These errors will be explained in the following paragraph.

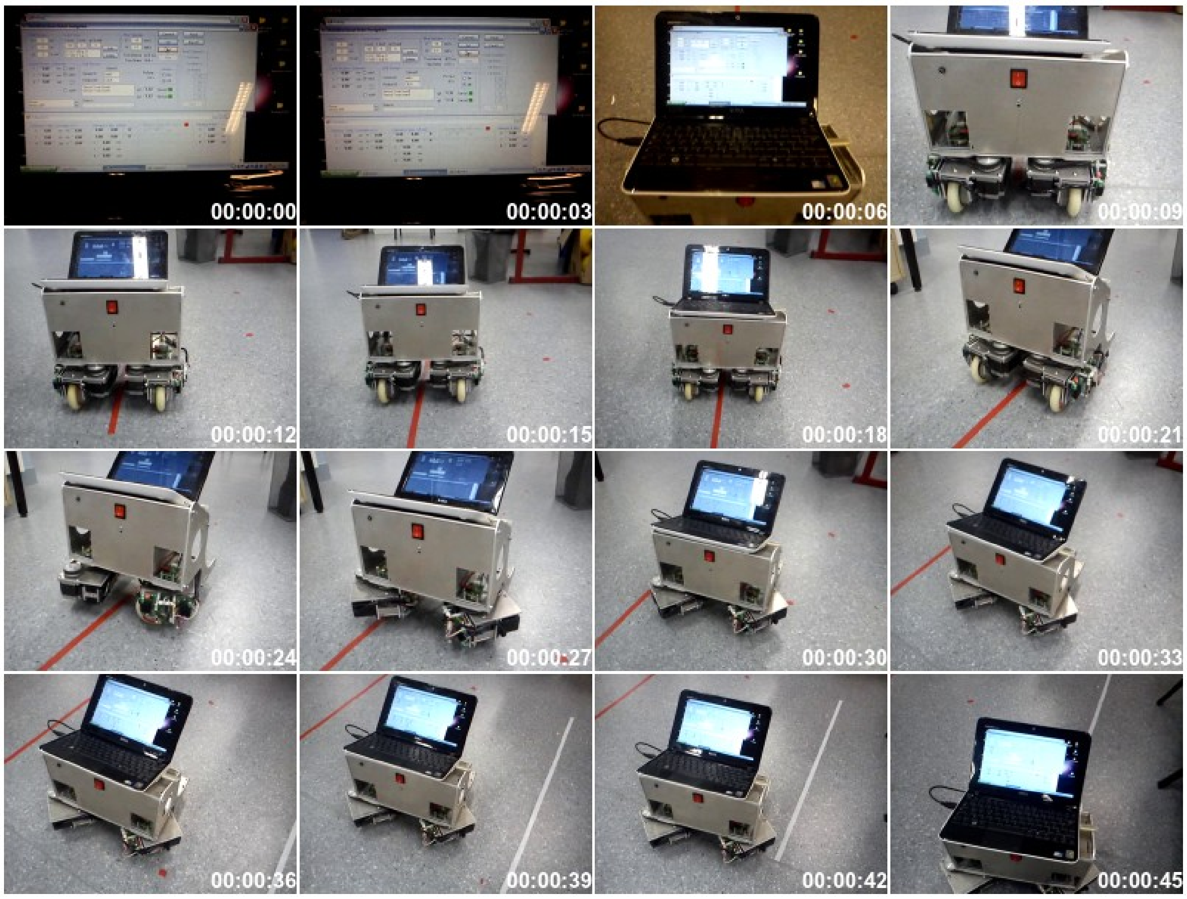


Fig 7.4 2nd Robot's Movement

7.2 Velocity – Odometry

A second cycle of experiments is needed to determine the accuracy of odometry and also the proper translation of the velocity calculated by the computer's program to frequency of the input signal of the motor's drivers. The experiments for measuring the accuracy of odometry was not completed by the end of this dissertation. Due to problems at the fabrication of the robot, the availability of absolute measuring system and the lack of time, we could not perform experiments to evaluate the methods used for odometry and the validity of the kinematic model.

However, during the first cycle of experiments we had the opportunity to view how the odometry functioned and what factors affected it. The methods of odometry we used, as explained in previous paragraphs, are actually numerical integration of either the velocities of each wheel separately or the velocities of the robot's centre. The numerical integration has as fundamental element the time interval Δt between each calculation. We measure this interval and we use it to complete the calculations. But, this implies that during this interval the robot or the motors moves at constant velocity. This is not true, due to hardware and mechanical reasons, and it is one of the most important factors that introduce errors in the calculation of odometry. A second factor is the assumption that the velocities of the wheels

($\dot{\theta}$) are respected. In reality, there is friction and slipping between the wheel and the floor or the expected acceleration is not met. This assumption is crucial for the localization of the robot using our methods of odometry and it makes them vulnerable to exterior interference and mechanical flaws.

The second part of this experiment was to calibrate the robots velocity, i.e. translation of the velocity calculated by the computer's program to frequency of the input signal of the motor's drivers. For example, the computer's program calculates a needed velocity of a joint at 1 rad/s. The micro-controller PIC who controls the specific motor of the joint has to produce a square wave to make the motor to rotate at 1 rad/s. So we have to find the relation between the rad/s calculated and the frequency of the square wave. The simplest way to achieve this was to perform a specific movement that was measured in advance. The movement was a forward movement of 1 metre. We marked 1 metre on the floor and we made the robot to move on the line. Each time, we multiplied the computed velocities with a coefficient and we noticed where the robot would stop. We corrected the coefficient accordingly and we repeated the movement, keeping every other factor constant (ie the angle of the wheels, initial point). After finding the coefficient that made the robot to move exactly 1 metre, we tried other maximum velocities to see if the coefficient would worked at all range of the computed velocities.

Chapter 8
Conclusion – Future Work

8.1 Conclusion

In this work, we researched the domain of mobile robotics and we dealt with various aspects of a robotic system, the theoretical formulation of kinematics models and path planning algorithms, the electronics and the hardware for motion control, the robot programming, communication and user interface. Each aspect plays its role for the control a robot. Particularly, we dealt with omni-directional robots. We found out what is an omni-directional robot and what wheels can give to robots omni-directional ability. We chose to deal with omni-directional robots with caster wheels for reasons explained above. We analysed the hardware design of the wheels and the means we used to be able to control them. This gave us the opportunity to get in touch with micro-controllers, sensors and electronics and ways to communicate among them. Lastly, we dealt with issues of programming computer and micro-controllers and developing a general system for controlling this kind of robots.

While researching the bibliography on omni-directional robots, we discovered that caster wheels provide us advantages such as immunity to road conditions and uneven floors. Enhancing our research on omni-directional robots with caster wheels we found kinematics models and localizing methods based on odometry. Watching the similarities between the kinematics model and also their versatility to adapt to several configurations we came up with the idea to developed a system that could adapt and control these robots. A system that would be independent of the configuration of the robot and implement these kinematics models and localizing methods. The system would consist not only of a computer program, but also of electronics, sensors, micro-controllers and motors. So, we were led to develop a protocol for the design of the wheels and for the communication of the micro-controllers with the computer. By respecting this protocol, we can develop many configurations of robots and be able to control them with the same system.

The system we developed is able to control many configurations of omni-directional robot with caster wheels with just a few modifications of the kinematic model. Having a system to control this kind of robots, we can develop robots with different objectives without worrying about their control. Like our case, the robot with the three wheels, which is used for conducting the experiments, is developed for transporting a human while the robot with the two wheels is used for collecting small items. Both robots can use our system for motion control (in the second case an other program is needed for controlling the collection of the items).

Certainly, further work is needed to improve the accuracy of the movement. The few experiments that were performed, validate the versatility of the system to control different configurations of robots but reveal also errors in the localization when performing a more advanced movement. While the system computes that the robot has reached the desired goal position, that is not true. The errors are noticeable even with a naked eye and without the need of a sophisticated measuring system. But the advantage of this system is that this is just a small component of the whole system and it is the only item that has to be modified and improved. The rest of the system will be left intact. Furthermore, this gives the opportunity for further research on specific components of the system, such as the motion planning algorithm, thus improving the system but without the need for developing a new one.

8.2 Future Work

Now that we have a complete system for controlling omni-directional robots with caster wheels we need to focus on finding better localization and trajectory planning algorithms. Even from the little experiments done to test the odometry methods that were developed in this work, it is clear that we have to search either to improve the accuracy of these methods or to find better, more independent ways for localization. For improving the accuracy of our methods, encoders will have to be installed on the wheels so we would have the real velocity of the wheels and not to assume it is same as the velocity computed by the program. Furthermore, the computer's program has many possibilities for improving. Different configurations of robots can be saved as files and the user can select which to load at the start of the program.

An other aspect that is worth researching is the enhancement of the kinematic model and the development of the dynamic model of the robot. The dynamic model will give us the ability to better control the robot, overtaking errors due to the slipping of the wheels.

The motion planning algorithm is a crucial part that has to be improved. Even thought the PVT algorithm is a good path generator, it consists of a list of points that it connects them with straight lines. To create a more complex movement, we would have to create a large list of points, close to each other. A path planning/following algorithm is more suitable for this purpose.

Currently, a visual localization system is developed, for localizing the robot using a camera and known visual landmarks. This method is entirely independent from the robot and its characteristics. It will provide to the robot coordinates that will be used to determine its movement, like odometry. But, this system has also limitations, as it needs a specific bounded workspace to function.

Bibliography

[Austr45] Austriamicrosystems, *"AS5045 angle sensor datasheet"*, Revision 1.7, http://www.austriamicrosystems.com

[Berk05] M. D. Berkemeier and L. Ma, "Discrete control for visual servoing the ODIS robot to parking lot lines", *Proc. of the 2005 IEEE Int. Conf. on Robotics and Automation*, pp. 3149-3154, 2005

[Camp87] G. Campion, G. Bastin and Br. D' AndrCa-Novel, "Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots", *IEEE Trans. on Robotics and Automation*, vol. 4, no. 2, pp. 281-340, 1987

[Jung08] E. Jung, H. Y. Lee, J. H. Lee, B.-J. Yi, W. K. Kim, and S. Yuta, "Navigation of an Omni-directional Mobile Robot with Active Caster Wheels", *Proc. of the 2008 IEEE Int. Conf. on Robotics and Automation*, Pasadena, pp. 1659-1665, 2008

[Lee05] J. H. Lee, S. Yuta, E. Koyanagi and B.-J. Yi, "Command system and motion control for caster-type omni-directional mobile robot", *Proc. of the 2005 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2714-2720, 2005

[Moore00] K. L. Moore and N. S. Flann, "A six-wheeled omnidiretional autonomous mobile robot", *IEEE Control System Magazine*, vol. 20, no. 6, pp. 53-66, 2000

[Mour06] G. Mourioux, C. Novales, G. Poisson and P. Vieyres, "Omni-directional robot with spherical orthogonal wheels: concepts and analyses", *Proc. of the 2006 IEEE Int. Conf. on Robotics and Automation*, Orlando, pp. 3374-3379, 2006

[Muir87] P.F. Muir and C.P. Neuman, "Kinematic Modeling for Feedback Control of an Omnidirectional Wheeled Mobile Robot", *Proc. of the 1987 IEEE Int. Conf. on Robotics and Automation*, Raleigh, North Carolina, pp. 1772-1778, 1987

[Pekm09] K. Pekmetzi, "Αρχιτεκτονική – Προγραμματισμός των μικροελεγκτών PIC", in *Συστήματα Μικροϋπολογιστών*, vol. 2, Chapter 5, Εκδόσεις Συμμετρία, Αθήνα, 2009

[PIC18] Microchip, *"PIC18F2455 / 2550 / 4455 / 4550 data sheet"*, 2009, http://www.microchip.com

[Pin94] F.G. Pin and S.M. Killough, "A new family of omnidirectional and holonomic wheeled platforms for mobile Robots", *IEEE Trans. on Robotics and Automation*, vol. 10, no. 4, pp. 480-489, 1994

[Pois01] G. Poisson, Y. Parmantier and C. Novales, "Modélisation cinématique d'un robot mobile omnidirectionnel à roues sphériques", *XVième Congrès Français de Mécanique*, Nancy, France, pp. 198-203, 2001

[Sopr57] Soprolec, *"57HSxx Series Hybrid Stepping Motors datasheet"*, http://www.soprolec.com

[Tech23] Techlf, *"23H056-x motor datasheet"*, http://www.techlf.com

[Tech8A] Techlf, *"SMD8A Microstep driver datasheet"*, 2007, http://www.techlf.com

[Ush03] N. Ushimi, M. Yamamoto, and A. Mohri, "Two wheels caster type odometer for omni-directional vehicles", *Proc. of the 2003 IEEE Int. Conf. on Robotics and Automation*, pp. 497-502, 2003

[Wada00] M. Wada, A. Takagi, and S. Mori, "Caster drive mechanism for holonomic and omnidirectional mobile platforms with no over constraint", *Proc. of the 2000 IEEE Int. Conf. on Robotics and Automation*, pp. 1531-1538, 2000

[West95] M. West and H. Asada, "Design and Control of Ball Wheel Omnidirectional Vehicles", *Proc. of the 1995 lEEE lnt. Conf. on Robotic and Automation*, pp. 1931-1938, 1995

[XBee00] Digi, *"XBee® Multipoint RF Modules"*, http://www.digi.com