2006

A Third-order Differential Steering Robot And Trajectory Generation In The Presence Of Moving Obstacles

Vatana An
University of Central Florida

A THIRD-ORDER DIFFERENTIAL STEERING ROBOT AND TRAJECTORY GENERATION IN THE PRESENCE OF MOVING OBSTACLES

by

VATANA AN
B.S.C.E. University of Florida, 2002
B.S.E.E. University of Florida, 2002

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2005

ABSTRACT

In this thesis, four robots will be used to implement a collision-free trajectory planning/replanning algorithm. The existence of a chained form transformation so that the robot's model can be control in canonical form will be analyzed and proved. A trajectory generation for obstacles avoidance will be derived, simulated, and implemented. A specific PC based control algorithm will be developed.

Chapter two describes two wheels differential drive robot modeling and existence of controllable canonical chained form. Chapter 3 describes criterion for avoiding dynamic objects, a feasible collision-free trjectory parameterization, and solution to steering velocity. Chapter 4 describes robot implementation, pc wireless interface, and strategy to send and receive information wirelessly. The main robot will be moving in a dynamically changing environment using canonical chained form. The other three robots will be used as moving obstacles that will move with known piecewise constant velocities, and therefore, with known trajectories. Their initial positions are assumed to be known as well. The main robot will receive command from the computer such as how fast to move and to turn in order to avoid collision. The robot will autonomously travel to the desired destination collision-free.

ACKNOWLEDGMENTS

I would like to express my thankfulness to my advisor, Dr. Zhihua Qu, for giving me suggestions, direction, and support, and my committee members, Dr. Wasfy Mikhael and Dr. Michael Haralambous, for their assistance and advice. I would like to thank the control and robotic lab fellows, Dr. Jing Wang, Mr. Jian Yang, and Mr. Ernesto Inoa for meaningful discussion on different research topics and applications. I also would like to thank my parents, my aunt, and my uncle for giving me constant financial support, spiritual support, and most important of all their unconditional love. Finally I would like to thank my brilliant sisters, Ratha, Borey, and Vithia, for their love and their encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In theory, a robot or other objects to be control can be assumed perfect. Realistically, even with today's reasonable price devices, a robot may not behave as well as desired. Typical problems include motor's slow rate of convergence, compass' and encoder's resolution, uneven weight, wear, rounding problem, and uneven floor. However, under empirical experimentations, a robot can be controlled to perform exceptionally well. In [4] and [5], a two-wheel robot was used to implement different control laws. Even when theoretical result guarantee some form of local asymptotic stability or convergence, practical implementations may show sign of oscillatory or even unstable behavior.

In this thesis, a two-wheel differential robot will be used to implement a trajectory generation in the presence of moving obstacles algorithm. A two-wheel differential drive robot to be controlled will be represented by a third order canonical chained form. A robot's trajectory can be generated between any two points as long as its boundary conditions and kinematic constraint are not violate. Obstacle avoidance techniques will be used to steer the robot so that collision-free trajectory can be achieved. Obstacles can be detect on the fly when they come within sensing range. However, for simplicity, the robot are assumed to know the paths of all the obstacles that come within sensing range.

The two wheels robot and moving obstacles will be wirelessly controlled from a PC using a Fast Wireless Communication Module (FWCM). A PC is the central processor that send/receive update information to/from the robot. All robot and obstacles have on board microcontroller (OOPIC-R), magnetic com-

pass, encoder, and FWCM.

2 THEORETICAL BACKGROUNDS

2.1 Introduction

Murray and Sastry introduced chained form in [3] to steer nonholonomic car-like robot from point A to point B. As will be seen in this chapter, chained form is highly nonlinear. Traditional control technique for linear system cannot be used to study chained form. Fortunately, chained form can be study by using nonlinear tool, Lie Bracket. A model on a two-wheel differential robot is the same as a four-wheel differential robot and will be derived and transformed into chained form. A few terms will be defined in the next section to study chained form.

2.2 Definition

The following terms are very important in studying chained form, which is highly nonlinear.

Controllability : Controllability refers to a system that has input $u(t)$ that is able to drive a system from an initial state to a desired state.

Diffeomorphism : A diffeomorphism is a map between manifolds which is differentiable and has a differentiable inverse.

Involutivity: A distribution is involutive if it is closed under Lie Bracket operations.

Lie Bracket : A Lie Bracket takes two n dimensional vectors and returns a new n-vector of linearly independent columns, it satisfies skew symmetry and Jacobi identity.

Linear Independence: A set of n vectors $v_1, v_2, ..., v_n$ is linearly independent if and only if the matrix rank of the matrix $\triangle_o = (v_1 v_2 ... v_n)$ is n.

Nonholonomic System : A nonholonomic system can instantly move forward and backward, but cannot move to the right or to the left without wheels slipping. To go to the right, the robot must turn right then drive forward.

Rank : The rank of any matrix A is the maximum number of linearly independent columns in the matrix.

Span: The span of a set of vector $v_1, v_2, ..., v_n$ is the set of all their linear combinations.

2.3 Modeling A Two-Wheel Differential Drive Robot

The complicity of robot modeling range from simple to complicate systems. A car with one trailer as in [10] is a fifth order kinematic model. A car-like robot as in [1] is a fourth order kinematic model. In this thesis a two-wheel differential drive robot is a third order kinematic model. Unlike modeling of a car-like robot, a two-wheel differential drive robot is very easy to model as seen in figure 1.

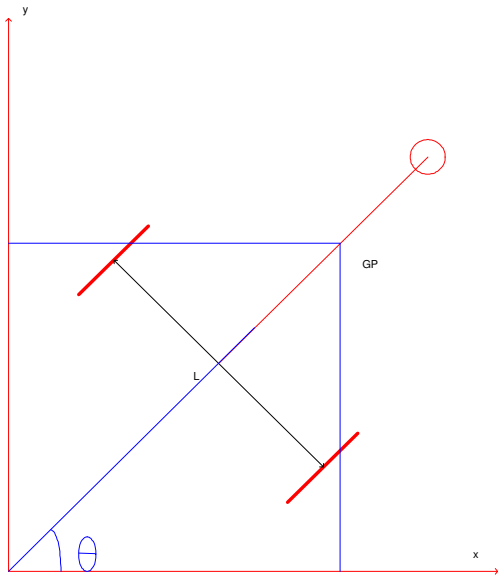

Figure 1: A simple model of two wheels robot

The following equations are obtained from figure 1:

$$\begin{aligned} \dot{x} &= u_1 cos(\theta), \\ \dot{y} &= u_1 sin(\theta), \\ \dot{\theta} &= u_2, \end{aligned} \tag{2.1}$$

where u_1 and u_2 represent speeding and turning respectively. Direct control on $\dot{x}$ can be done by letting $v_{c1} = u_1 cos(\theta)$, and u_2 is to be determined. Equations in (2.1) can be rewritten as

$$\begin{aligned} \dot{x} &= v_{c1}, \\ \dot{y} &= v_{c1} tan\theta, \\ \dot{\theta} &= u_2. \end{aligned} \tag{2.2}$$

To obtain a canonical chained form, let $z_3 = y$. Section 2.5 will show why $z_3 = y$ is chosen to begin with. Taking derivative on both sides yield

$$\dot{z}_3 = v_{c1} tan\theta. \tag{2.3}$$

But $\dot{z}_3 = v_{c1} z_2$ according to a chained form format that will be discussed in the next section, section 2.4. Therefore z_2 is equal to $tan\theta$ in (2.3). The sufficient conditions that the assumption can be make will be proved in the next section. Taking derivative of z_2 yield

$$\begin{aligned} \dot{z}_2 &= sec^2\theta\dot{\theta}, \\ \dot{z}_2 &= sec^2\theta u_2, \end{aligned}$$

and by letting $v_{c2} = sec^2\theta u_2$, the following transformation will yield the canonical chained form:

$$\begin{aligned} z_1 &= x, \\ z_2 &= tan\theta, \\ z_3 &= y, \\ u_1 &= v_{c1} sec\theta, \\ u_2 &= v_{c2} cos^2\theta. \end{aligned} \tag{2.4}$$

Replacing u_2 in term of v_{c2} in (2.2) yields

$$\begin{aligned} \dot{x} &= v_{c1}, \\ \dot{y} &= v_{c1} tan\theta, \\ \dot{\theta} &= v_{c2} cos^2\theta. \end{aligned} \tag{2.5}$$

Equations above can be transform into the following form, the third order chained form:

$$\begin{aligned} \dot{z}_1 &= v_{c1}, \\ \dot{z}_2 &= v_{c2}, \\ \dot{z}_3 &= z_2 v_{c1}. \end{aligned} \tag{2.6}$$

2.4 Chained Form

Chained form is a nilpotent form that can be used as a canonical form to describe nonholonomic system. Many mechanical systems with nonholonomic constraints can be locally or globally converted to chained form through coordinate change and control mapping. A canonical form allows one to design open-loop or closed-loop controls for a whole class of nonholonomic systems. A closed form kinematic model of chained form mobile robot is very useful for kinematic analysis, design, and modeling of similar structures.

A chained form, a special triangular form, for n order

$$\begin{aligned} \dot{z}_1 &= v_{c1}, \\ \dot{z}_2 &= v_{c2}, \\ \dot{z}_3 &= z_2 v_{c1}, \\ &\vdots \\ \dot{z}_n &= z_{n-1} v_{c1}, \end{aligned} \tag{2.7}$$

can be obtained from system of the form

$$\dot{\zeta} = g_1(\zeta)u_1 + g_2(\zeta)u_2. \tag{2.8}$$

The requirement is that (2.8) must be locally feedback transformable given

$$\begin{aligned} \triangle_o &:= span\{g_1, g_2, ad_{g1}g_2, \cdots, ad_{g1}^{n-2}g_2\}, \\ \triangle_1 &:= span\{g_2, ad_{g1}g_2, \cdots, ad_{g1}^{n-2}g_2\}, \\ \triangle_2 &:= span\{g_2, ad_{g1}g_2, \cdots, ad_{g1}^{n-3}g_2\}. \end{aligned} \tag{2.9}$$

$\triangle_o(\zeta) = \Re^n$ for all ζ in open set $U \subset \Re^n$ and $\triangle_1$ is involutive on U. A locally feedback transformation will be in the following form:

$$\begin{array}{rcl} z & = & \phi(\zeta) \\ v_c & = & \beta(\zeta, u). \end{array} \tag{2.10}$$

Note that equations (2.4) are in the same form as equations (2.10). Rewriting (2.5) in the following form

$$\dot{\zeta}(x, y, \theta, u) = \begin{pmatrix} 1 \\ tan\theta \\ 0 \end{pmatrix} v_{c1} + \begin{pmatrix} 0 \\ 0 \\ cos^2\theta \end{pmatrix} v_{c2}, \tag{2.11}$$

a systematic proof can be done to show that it can be locally feedback transform into a third order chained form.

2.5 Proof

If Δ_1 is an involutive distribution of dimension n-2, there exists a function h such that $dh \cdot \Delta_2 = 0$ and $dh \cdot ad_{g_1}^{n-2} g_2 \neq 0$ that map $\zeta \rightarrow z$ as

$$\begin{array}{rcl} z_1 & = & x, \\ z_2 & = & L_{g_1}^{n-2} h \\ & \cdot & \\ & \cdot & \\ & \cdot & \\ z_n & = & y. \end{array}$$

Since $n = 3$ for a third order system, z_3 is chosen to be y as suggested under equations (2.2). Using Lie Bracket, $L_{g1} g_2$ is obtained as

$$\begin{array}{rcl} L_{g_1} g_2 & = & \dfrac{\partial g_2}{\partial \zeta} g_1 - \dfrac{\partial g_1}{\partial \zeta} g_2, \\ L_{g_1} g_2 & = & \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}. \end{array} \tag{2.12}$$

From (2.8), (2.9), and (2.12)

$$\begin{aligned}
\Delta_o &= \begin{pmatrix} 1 & 0 & 0 \\ tan\theta & 0 & -1 \\ 0 & cos^2\theta & 0 \end{pmatrix}, \\
\Delta_1 &= \begin{pmatrix} 0 & 0 \\ 0 & -1 \\ cos^2\theta & 0 \end{pmatrix}, \\
\Delta_2 &= \begin{pmatrix} 0 \\ 0 \\ cos^2\theta \end{pmatrix}.
\end{aligned}$$

If $h_1 = x_1$ is chosen, then

$$\begin{aligned}
dh_1 \cdot \Delta_1 &= 0, \\
dh_1 \cdot g_1 &= 1, \\
dh_1 \cdot \Delta_2 &= 0, \\
dh_2 \cdot \Delta_2 &= 0, \\
dh_2 \cdot ad_{g_1}^{n-2} g_2 &\neq 0,
\end{aligned}$$

ensure that there exists a locally feedback transformation.

3 TRAJECTORY GENERATION

3.1 Introduction

In [1] the authors stated that a trajectory is *feasible* if the boundary conditions and kinematic constraints are satisfy. In that paper, a fourth order chained form was used to study in determining trajectories in the presence of boundary constraints, kinematic constraints, and moving obstacles. In this thesis, the same idea will be applying on a third order chained form.

Lemma 1: For the kinematic model in chained form (2.6), there exist two inputs v_{c1} and v_{c2}. To obtain a feasible trajectory $z_3 = F(z_1)$ between two boundary conditions $z(t_o) = z^o = [z_1^o, z_2^o, z_3^o]^T$ and $z(t_f) = z^f = [z_1^f, z_2^f, z_3^f]^T$, the following must hold:

$$z_1^o \neq z_1^f.$$

If $z_1^o = z_1^f$, there must be an intermediate point z^m with $z^m \neq z_1^o = z_1^f$ so that singularity can be avoided. From (2.4), the boundary conditions in chained form are

$$z_1^o = x^o, \quad z_3^o = F(z_1^o) = y^o, \quad \left.\frac{dz_3}{dz_1}\right|_{z_1=z_1^o} = tan(\theta_o), \tag{3.1}$$

$$z_1^f = x^f, \quad z_3^f = F(z_1^f) = y^f, \quad \left.\frac{dz_3}{dz_1}\right|_{z_1=z_1^f} = tan(\theta_f). \tag{3.2}$$

3.2 Criterion For Avoiding Dynamic Objects

Consider time interval $t \in [t_o+kT_s, t_o+(k+1)T_s]$ that the robot is at coordinate $(x(t), y(t))$ and the *ith* obstacle is at coordinate $(x_i(t), y_i(t))$ in the figure below. The robot is moving with to be determined vector velocity $v_r \triangleq [\dot{x}(t)\ \dot{y}(t)]^T$.

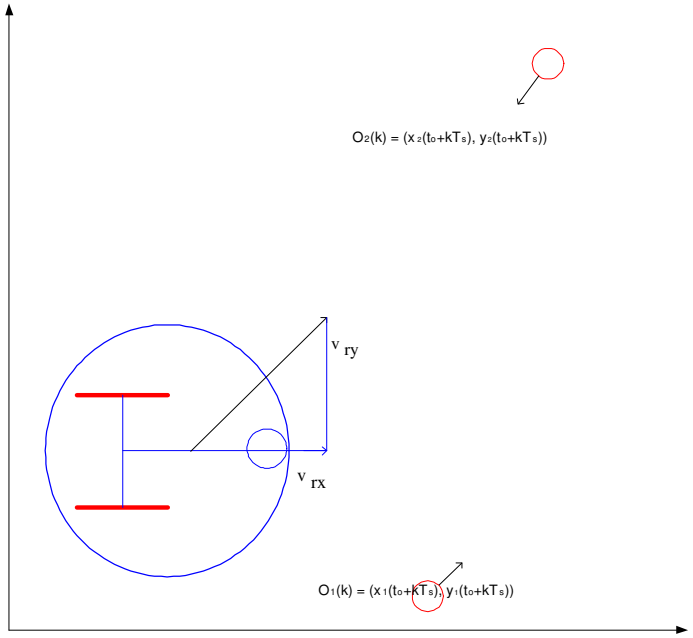

Figure 2: A robot in the presence of moving obstacles

The time-varying obstacle can be model as a moving point at initial location $O_i = (x_i^k, y_i^k)$ with radius r, where $x_i^k = x_i(t_o + kT_s)$ and $y_i^k = y_i(t_o + kT_s)$. The point O_i is moving at a known constant velocity $v_i^k \triangleq [v_{i,x}^k\ v_{i,y}^k]^T$.

The relative velocity between the robot and the *ith* obstacle is defined as

$$v_{r,i}^k \triangleq v_r - v_i^k. \tag{3.3}$$

Considering relative velocity concept, figure 2 can be view as figure 3 in which obstacles are "static."

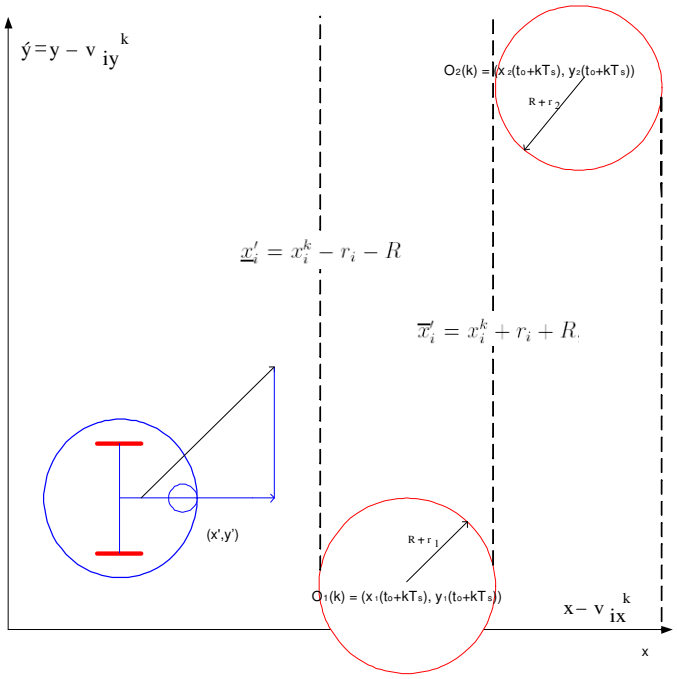

Figure 3: A robot in the presence of "static" obstacles

Clearly seen from figure 3, there will be no collision for $x'_i \in [\underline{x}'_i, \overline{x}'_i]$ if

$$(y'_i - y_i^k)^2 + (x'_i - x_i^k)^2 \geq (r_i + R)^2, \tag{3.4}$$

where $\underline{x}'_i = x_i^k - r_i - R$, $\overline{x}'_i = x_i^k + r_i + R$, $x'_i = x - v_{i,x}^k \tau$, $y'_i = y - v_{i,y}^k \tau$, and $\tau = t - (t_o + kT_s)$ for $t \in [t_o + kT_s, t_o + T]$. In term of state transformation (2.4), (3.4) become

$$(z'_{3,i} - y_i^k)^2 + (z'_{1,i} - x_i^k)^2 \geq (r_i + R)^2, \tag{3.5}$$

where $z'_{1,i} = z_1 - v_{i,x}^k \tau$ and $z'_{3,i} = z_3 - v_{i,y}^k \tau$.

3.3 A Feasible Collision-Free Trajectory Parameterization

The time when a robot first start to move can assume to be 0. So for the first sampling instant, the robot move from 0 to kT_s. Using the first equation in

(2.6) where $dz_1 = v_{c1}dt$, z_1 can be find by integrating both sides:

$$\int_0^{kT_s} dz_1 = \int_0^{kT_s} v_{c1}dt.$$

Letting $z_i^k = z_i(t_o + kT_s)$ and $z_i^{k-1} = z_i(t_o + (k-1)T_s)$, the above equation become

$$\begin{aligned} z_1^k &= z_1^o + k\frac{z_1^f - z_1^o}{\bar{k}}, \\ z_1(t) &= z_1^k + \frac{z_1^f - z_1^o}{T}(t - t_o - kT_s) \quad \forall t \quad \in [t_o + kT_s, t_f]. \end{aligned}$$

From the second equation in (2.6)

$$\begin{aligned} \int_{t_o+(k-1)T_s}^{t_o+kT_s} dz_2 &= \int_{t_o+(k-1)T_s}^{t_o+kT_s} v_{c2}dt, \\ z_2^k - z_2^{k-1} &= \int_{t_o+(k-1)T_s}^{t_o+kT_s} v_{c2}dt, \\ z_2^k &= z_2^{k-1} + \int_{t_o+(k-1)T_s}^{t_o+kT_s} v_{c2}dt. \end{aligned}$$

From the third equation in (2.6)

$$\begin{aligned} \frac{dz_3}{dt} &= z_2 v_{c1}, \\ z_3^k &= z_3^{k-1} + v_{c1}\int_{t_o+(k-1)T_s}^{t_o+kT_s} z_2^k ds. \end{aligned} \tag{3.6}$$

Replacing v_{c1} and z_2^k from above,

$$\begin{aligned} z_3^k &= z_3^{k-1} + v_{c1}z_2^{k-1}(t_o + kT_s - (t_o + (k-1)T_s)) + v_{c1}\int_{t_o+(k-1)T_s}^{t_o+kT_s}\int_{t_o+(k-1)T_s}^{s} v_{c2}^{k-1}dsdt, \\ z_3^k &= z_3^{k-1} + v_{c1}z_2^{k-1}T_s + v_{c1}\int_{t_o+(k-1)T_s}^{t_o+kT_s}\int_{t_o+(k-1)T_s}^{s} v_{c2}^{k-1}dsdt. \end{aligned} \tag{3.7}$$

Following the same proposition as in [1], a class of feasible and collision-free trajectories can be parameterized in polynomial and matric form as

$$z_3^k(z_1) = a_o^k + a_1^k z_1 + a_2^k z_1^2 + a_3^k z_1^3 + a_4^k z_1^4, \tag{3.8}$$

$$z_3(z_1) = F(z_1) = a^k f(z_1), \tag{3.9}$$

respectively, where a^k is a constant vector to be determined, and $f(z_1)$ is a vector composed of basis function $z_1(t)$:

$$\begin{aligned} a^k &= [a_o^k, a_1^k, a_2^k, a_3^k, a_4^k], \\ f(z_1) &= [1, z_1(t), (z_1(t))^2, (z_1(t))^3, (z_1(t))^4]^T. \end{aligned}$$

Taking derivative of (2.6) with respect to z_1 yields

$$\frac{dz_3^k}{dz_1^k} = a_1^k + 2a_2^k z_1^k + 3a_3^k (z_1^k)^2 + 4a_4^k (z_1^k)^3, \tag{3.10}$$

where dz_3^k/dz_1^k can be found from (2.5) as

$$\begin{aligned} \frac{dz_3^k}{dz_1^k} &= \frac{z_2^k v_{c1}^k}{v_{c1}^k}, \\ \frac{dz_3^k}{dz_1^k} &= z_2^k, \\ z_2^k &= a_1^k + 2a_2^k z_1^k + 3a_3^k (z_1^k)^2 + 4a_4^k (z_1^k)^3. \end{aligned} \tag{3.11}$$

From (3.1) and (3.2), the initial and the final boundary conditions are given as follow

$$\begin{aligned} z_3^k &= a_o^k + a_1^k z_1^k + a_2^k (z_1^k)^2 + a_3^k (z_1^k)^3 + a_4^k (z_1^k)^4, \\ z_2^k &= a_1^k + 2a_2^k z_1^k + 3a_3^k (z_1^k)^2 + 4a_4^k (z_1^k)^3, \\ z_3^f &= a_o^k + a_1^k (z_1^f) + a_2^k (z_1^f)^2 + a_3^k (z_1^f)^3 + a_4^k (z_1^f)^4, \\ z_2^f &= a_1^k + 2a_2^k z_1^f + 3a_3^k (z_1^f)^2 + 4a_4^k (z_1^f)^3. \end{aligned} \tag{3.12}$$

When a_4^k is determined, the remaining a coefficients can be found from the four equations four unknown variables in (3.12). Matrically,

$$Y^k = (B^k)[a_o^k, a_1^k, a_2^k, a_3^k]^T + A^k a_4^k, \tag{3.13}$$

where

$$Y^k = \begin{bmatrix} z_3^k \\ z_2^k \\ z_3^f \\ z_2^f \end{bmatrix}, \quad A^k = \begin{bmatrix} {z_1^k}^4 \\ 4(z_1^k)^3 \\ {z_1^f}^4 \\ 4(z_1^f)^3 \end{bmatrix},$$

and

$$B^k = \begin{bmatrix} 1 & z_1^k & (z_1^k)^2 & (z_1^k)^3 \\ 0 & 1 & 2z_1^k & 3(z_1^k)^2 \\ 1 & z_1^f & (z_1^f)^2 & (z_1^f)^3 \\ 0 & 1 & 2z_1^f & 3(z_1^f)^2 \end{bmatrix}.$$

Solving for $[a_o^k, a_1^k, a_2^k, a_3^k]^T$ in (3.12), we have

$$[a_o^k, a_1^k, a_2^k, a_3^k]^T = (B^k)^{-1}(Y^k - A^k a_4^k). \tag{3.14}$$

3.4 Solution To Steering Velocity

From (3.5)

$$(z_3 - v_{i,y}^k \tau - y_i^k)^2 + (z_1 - v_{i,x}^k \tau' - x_i^k)^2 \geq (r_i + R)^2. \tag{3.15}$$

The key to simplify the above equation is by simplifying z_3. Rewriting (3.8) as

$$z_3^k(z_1) = \begin{bmatrix} 1 & z_1 & z_1^2 & z_1^3 \end{bmatrix} \begin{bmatrix} a_o^k \\ a_1^k \\ a_2^k \\ a_3^k \end{bmatrix} + a_4^k z_1^4. \tag{3.16}$$

Replacing (3.14) in the above equation, we have

$$z_3^k(z_1) = \underline{f}(z_1)(B^k)^{-1}(Y^k - A^k a_4^k) + a_4^k z_1^4.$$

Or

$$z_3^k(z_1) = \underline{f}(z_1)(B^k)^{-1}Y^k + (z_1^4 - \underline{f}(z_1)(B^k)^{-1}A^k)a_4^k. \tag{3.17}$$

Where $\underline{f}(z_1) = \begin{bmatrix} 1 & z_1 & z_1^2 & z_1^3 \end{bmatrix}$. Replacing (3.17) in (3.15) yields a second order polynomial

$$\min_{t \in [\underline{t}_i^*, \bar{t}_i^*]} g_2(z_1(t), k)(a_4^k)^2 + g_{1,i}(z_1(t), k, \tau)a_4^k + g_{0,i}(z_1(t), k, \tau)\Big|_{\tau = t - t_0 - kT_s} \geq 0, \tag{3.18}$$

where

$$
\begin{aligned}
g_2(z_1(t),k) &= \left[(z_1(t))^4 - \underline{f}(z_1(t))(B^k)^{-1}A^k\right]^2, \\
g_{1,i}(z_1(t),k,\tau) &= 2\left[(z_1(t))^4 - \underline{f}(z_1(t))(B^k)^{-1}A^k\right]\left[\underline{f}(z_1(t))(B^k)^{-1}Y^k - y_i^k - v_{i,y}^k\tau\right], \\
g_{0,i}(z_1(t),k,\tau) &= \left[\underline{f}(z_1(t))(B^k)^{-1}Y^k - y_i^k - v_{i,y}^k\tau\right]^2 + (z_1(t) - x_i^k - v_{i,x}^k\tau)^2 - (r_i+R)^2.
\end{aligned}
$$

The next step is to determine the steering control input v_{c2}. $v_{c1}^k = C$ is assumed to be a known constant, although it can be made varying as long as it does not violate integration rule. Let $v_{c2}^k = C_o^k + C_1^k(t-t_o-kT_s) + C_2^k(t-t_o-kT_s)^2$, where $C_i^k, i = 0,1,2$ are constants. Directly integrating (2.6) yields equations that will lead to solution for steering velocity:

$$
\begin{aligned}
z_1(t) &= z_1^k + C(t-t_o-kT_s), \\
z_2(t) &= z_2^k + C_o(t-t_o-kT_s) + \frac{C_1}{2}(t-t_o-kT_s)^2 + \frac{C_2}{3}(t-t_o-kT_s)^3, \\
z_3(t) &= z_3^k + Cz_2^k(t-t_o-kT_s) + \frac{C_oC}{2}(t-t_o-kT_s)^2 + \frac{C_1C}{6}(t-t_o-kT_s)^3 \\
&\quad + \frac{C_2C}{12}(t-t_o-kT_s)^4, \qquad (3.19)
\end{aligned}
$$

for $t \in (t_o+kT_s, t_o+(k+1)T_s]$. Substituting $z_1(t) = z_1^k + C(t-t_o-kT_s)$ into $z_3 = a^k f(z_1)$ yields

$$
\begin{aligned}
z_3(t) &= b_o + b_1(t-t_o-kT_s) + b_2(t-t_o-kT_s)^2 \\
&\quad + b_3(t-t_o-kT_s)^3 + b_4(t-t_o-kT_s)^4,
\end{aligned}
$$

where

$$
\begin{aligned}
b_o &= a_o^k + a_1^k z_1^k + a_2^k(z_1^k)^2 + a_3^k(z_1^k)^3 + a_4^k(z_1^k)^4, \\
b_1 &= a_1^k C + 2a_2^k z_1^k C + 3a_3^k(z_1^k)^2 C + 4a_4^k(z_1^k)^3 C, \\
b_2 &= a_2^k C^2 + 3a_3^k z_1^k C^2 + 6a_4^k(z_1^k)^2 C^2, \\
b_3 &= 4a_4^k z_1^k C^3 + a_3^k C^3, \\
b_4 &= a_4^k C^4. \qquad (3.20)
\end{aligned}
$$

Comparing (3.19) and (3.20), the followings are obtained:

$$\begin{aligned} C_o^k &= 2a_2^k C + 6a_3^k z_1^k C + 12a_4^k (z_1^k)^2 C, \\ C_1^k &= 6a_3^k C^2 + 24a_4^k z_1^k C^2, \\ C_2^k &= 12a_4^k C^3. \end{aligned} \tag{3.21}$$

Equations above result in steering inputs to achieve obstacle avoidance path (3.9). For $t \in (t_o + kT_s, (k+1)T_s]$,

$$\begin{aligned} v_{c1}^k(t) &= \frac{z_1^f - z_1^o}{T}, \\ v_{c2}^k(t) &= (2a_2^k + 6a_3^k z_1^k + 12a_4^k (z_1^k)^2)C \\ &\quad +(6a_3^k + 24a_4^k z_1^k)(t - t_o - kT_s)C^2 \\ &\quad +12a_4^k (t - t_o - kT_s)^2 C^3. \end{aligned}$$

which result in the real and steering velocities of u_1 and u_2 respectively.

Based on the observation from the figure below, we can use u_1 and u_2 to find the speed of left and right wheels of the robot.

$$\dot{\theta} = (V_R - V_L)/L. \tag{3.22}$$

Since the guide point is at the center of the robot,

$$u_1 = (V_R + V_L)/2. \tag{3.23}$$

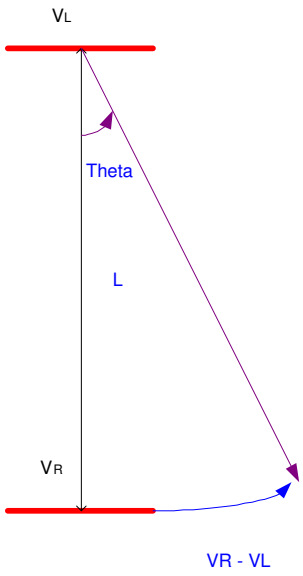

Figure 4: Wheels at different velocities

3.5 Simulation

The second order inequality polynomial that dictate the solution of a_4 has three different scenarios: both a_{4min} and a_{4max} are positive, both a_{4min} and a_{4max} are negative, and both are of different signs. When both a_{4min} and a_{4max} are positive or both are negative, $a_4(k)$ can be chosen to be zero. When both have different signs, $a_4(k)$ is the one with smaller magnitude. Appendix A contains code that can easily be changed to different dynamic setting. The code in appendix A follows the methodology displayed in figure 4.

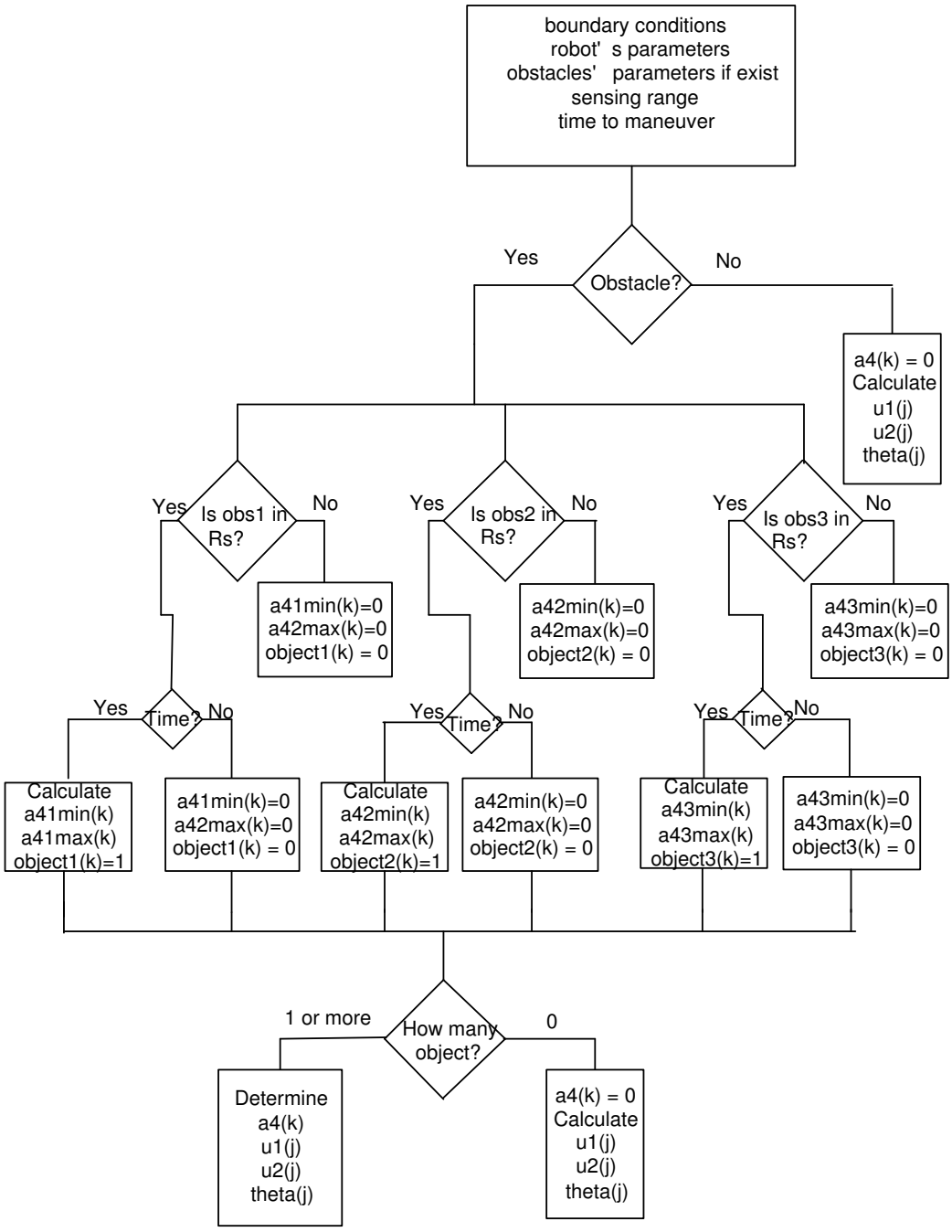

Figure 5: Determining and Calculating a4, u1, u2, and theta

For simplicity, all moving obstacles are assume to start from fixed locations as shown in the figure below:

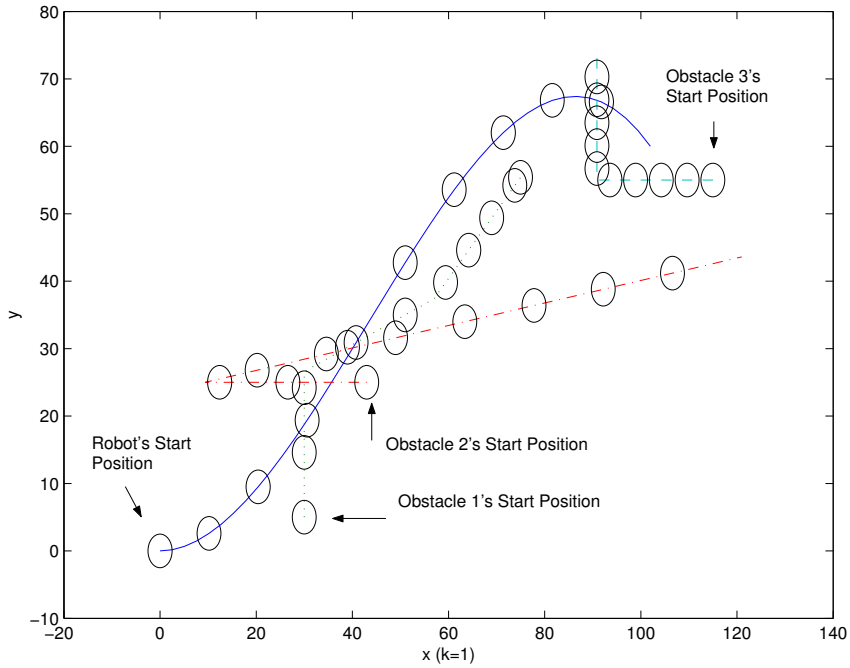

Figure 6: Robot and moving obstacles' trajectories

The trajectory of the robot is denoted by a solid blue line. The trajectory of the first obstacle is denoted by a dotted light-blue line. The trajectory of the second obstacle is denoted by a dash red line. The trajectory of the third obstacle is denoted by dash light-blue line. Along each trajectory, there exist 10 circles representing the robot's or obstacle's position during that sampling time. Placement of obstacles and robot was designed so that different interesting cases are shown. Below are the setting used to obtained the figure above:

oRobot Parameter: R = 5

oBoundary Condition: $q^0 = (0, 0, 0)$ and $q^f = (70, 120, pi/4)$

oMoving Obstacles: n = 3

$O_1(t_o) = [0, 20]^T$, $O_2(t_o) = [63, 35]^T$, $O_3(t_o) = [31, 60]^T$, and $r_i = 5$ for $i = 1$, 2, 3.

oDesign Parameters: $t_o = 0$, $T = 40$ seconds, and $T_s = 3$ seconds.

oSpeed of Obstacles:

$v_1^0 = v_1^1 = v_1^2 = [0, 2.4]^T$, $v_1^3 = v_1^4 = v_1^5 = [3, 1.2]^T$, $v_1^6 = v_1^7 = v_1^8 = v_1^9 = v_1^{10} = [1.2, 1.2]^T$, and $v_1^{11} = [0, 0]^T$,

$v_2^0 = v_2^1 = [-4.1, 0]^T$, $v_2^2 = [-3, 0]^T$, and $v_2^3 = v_2^4 = v_2^5 = v_2^6 = v_2^7 = v_2^8 = v_2^9 = v_2^{10} = v_2^{11} = [3.6, 0.6]^T$,

$v_3^0 = v_3^1 = v_3^2 = v_3^3 = v_3^4 = v_3^5 = [-1.34, 0]^T$, and $v_3^6 = v_3^7 = v_3^8 = v_3^9 = v_3^{10} = v_3^{11} = [0, 0.85]^T$.

oThe Solution to the parameterized trajectory is listed in table 1.

Observing the figure above which illustrated the first sampling instant, the robot would collide with obstacle one at time $k = 5$. The following two figures still illustrate the collision between the robot and obstacle one. At $k = 4$, when the robot detect that collision will occur, the algorithm generate a new trajectory to avoid collision. See the figure at $k = 4$.

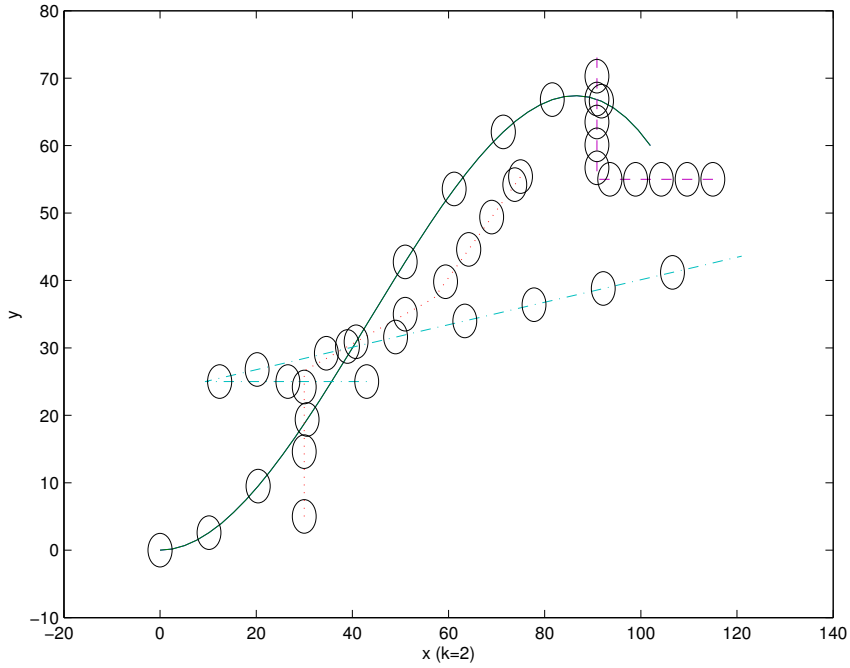

Figure 7: Robot's trajectory for the second sampling instant

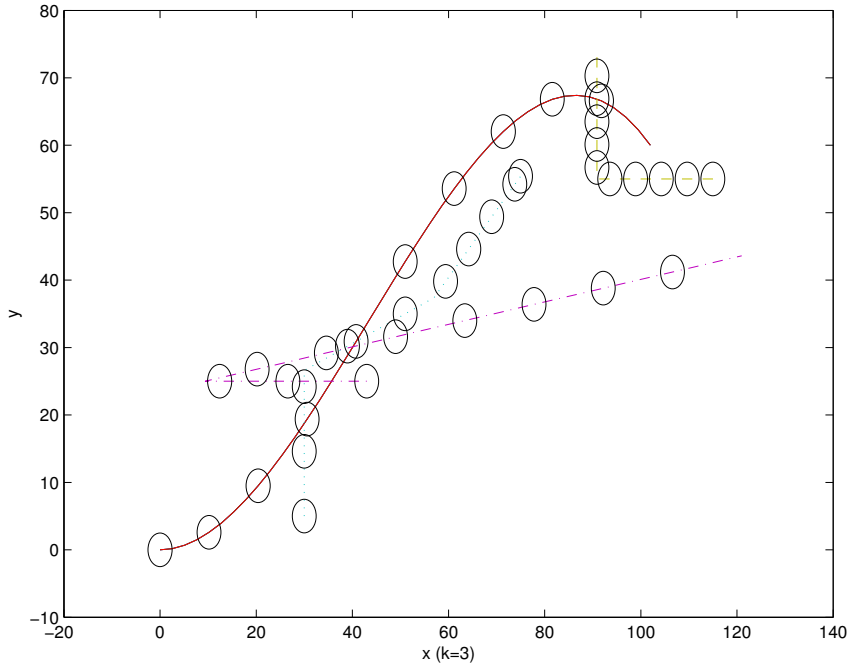

Figure 8: Robot's trajectory for the third sampling instant

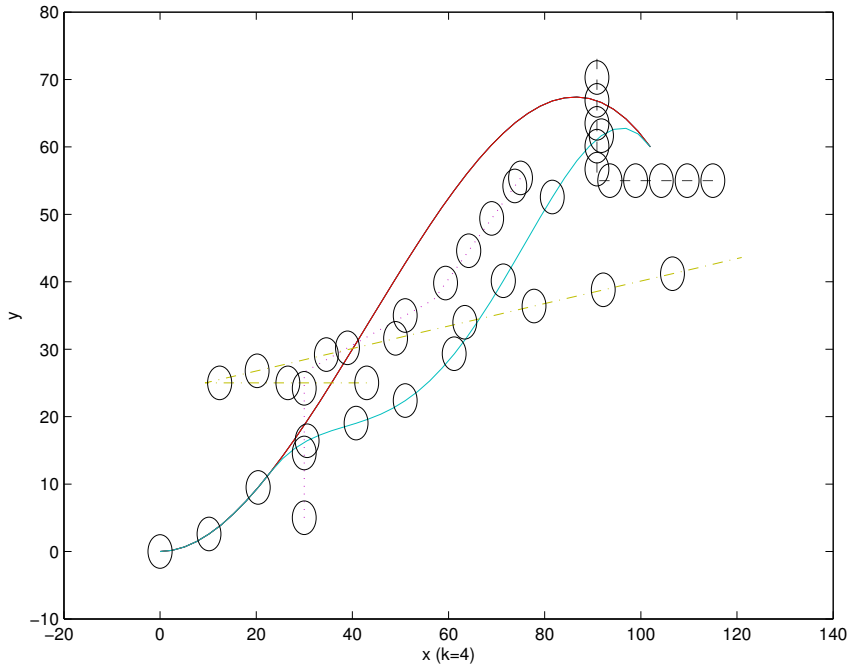

Figure 9: Robot's trajectory for the fourth sampling instant

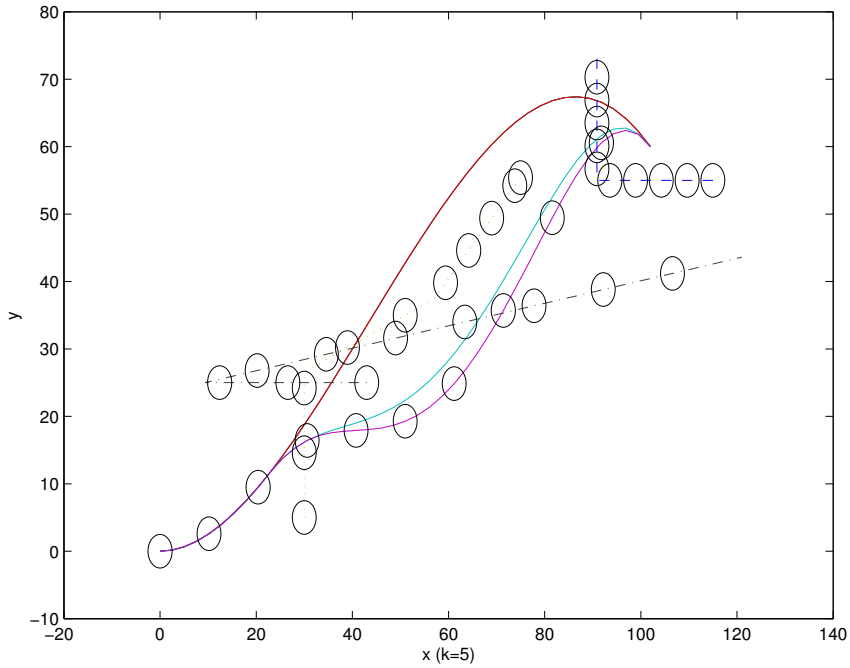

Figure 10: Robot's trajectory for the fifth sampling instant

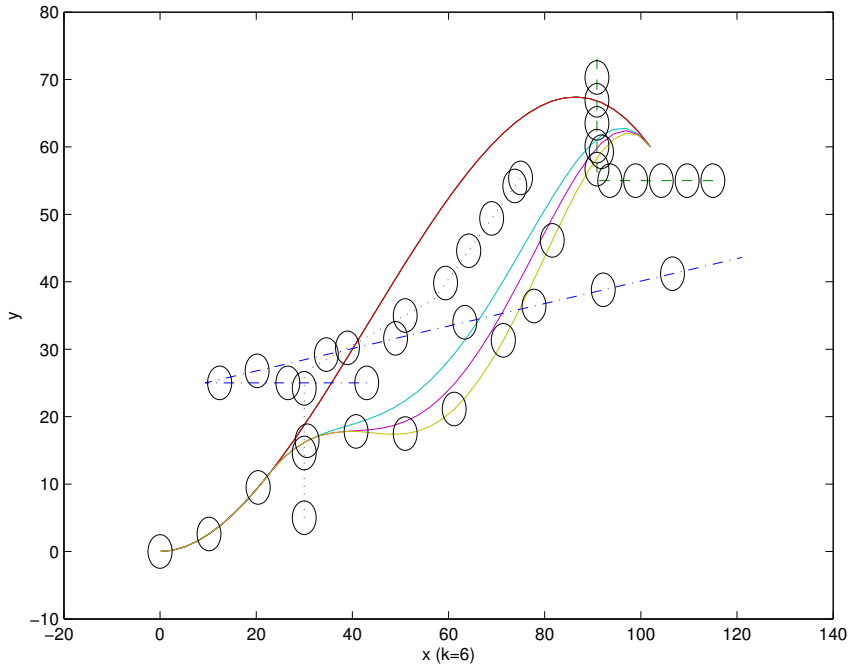

Figure 11: Robot's trajectory for the sixth sampling instant

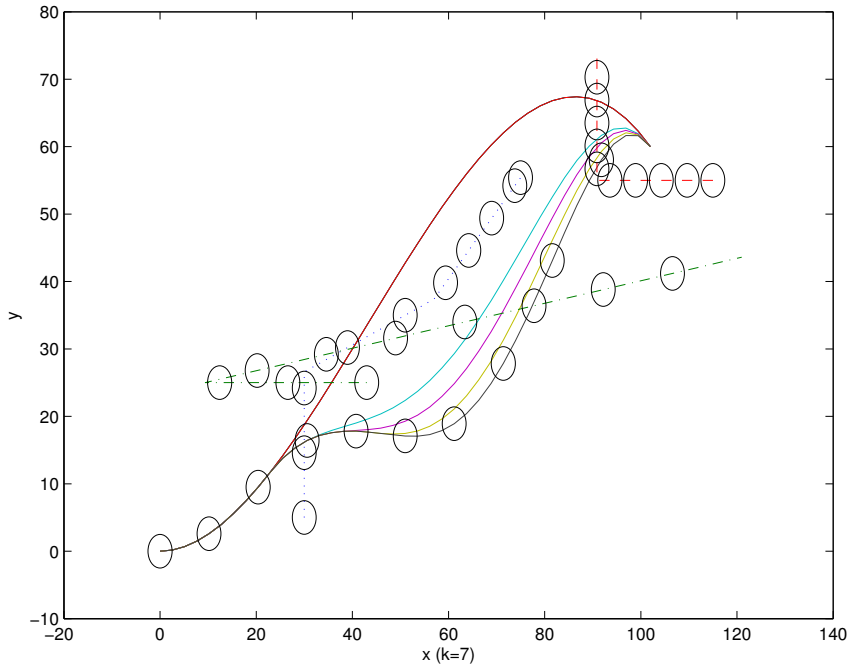

Figure 12: Robot's trajectory for the seventh sampling instant

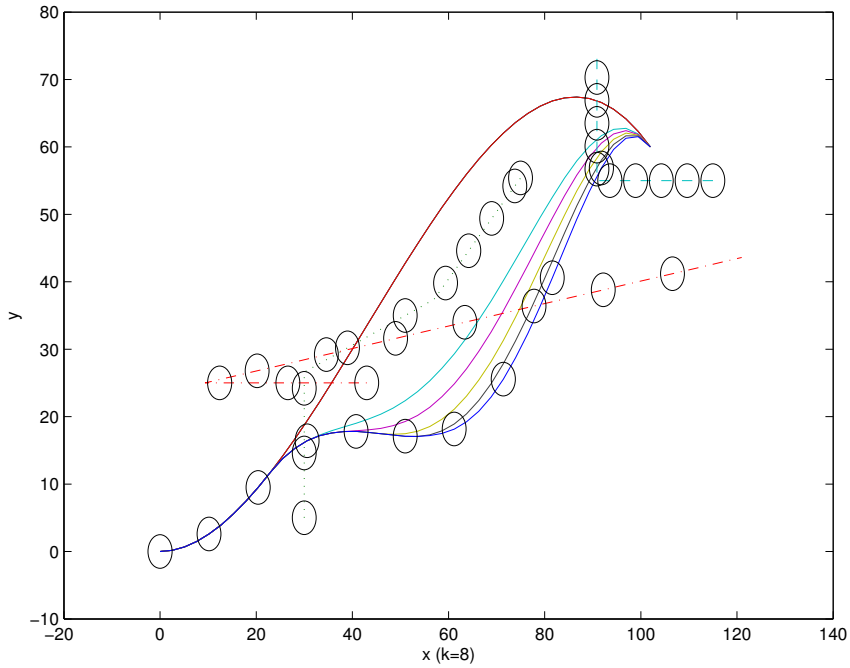

Figure 13: Robot's trajectory for the eighth sampling instant

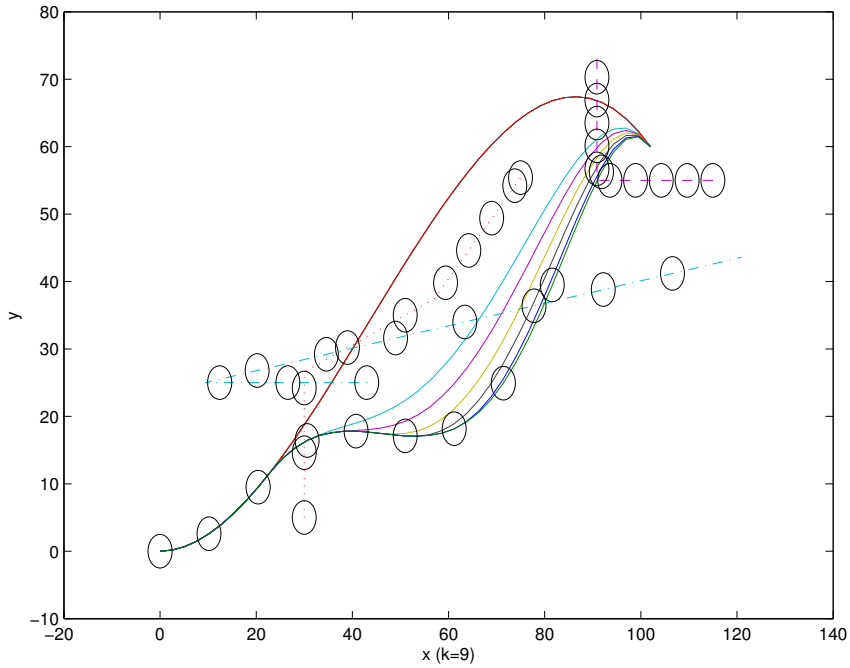

Figure 14: Robot's trajectory for the ninth sampling instant

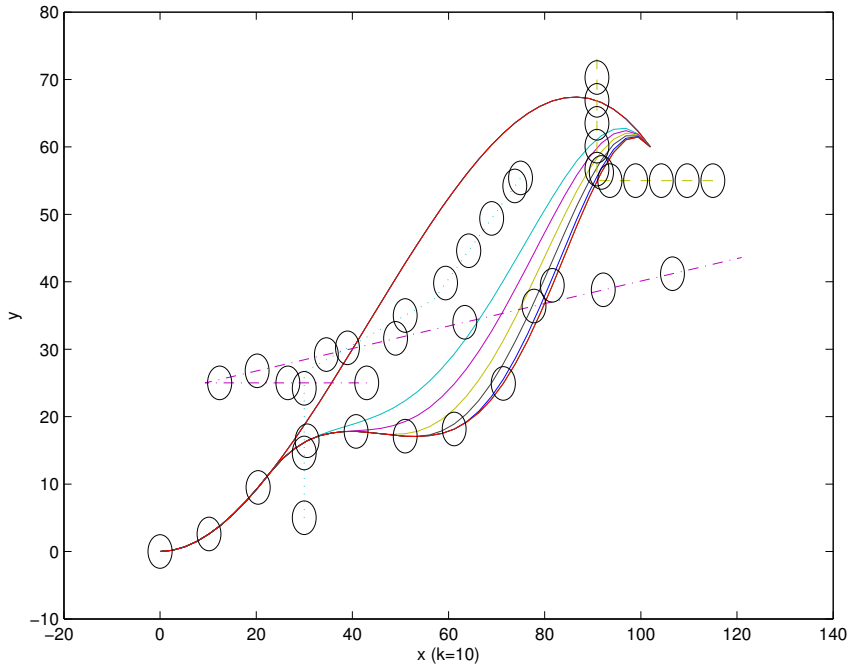

Figure 15: Robot's trajectory for the tenth sampling instant

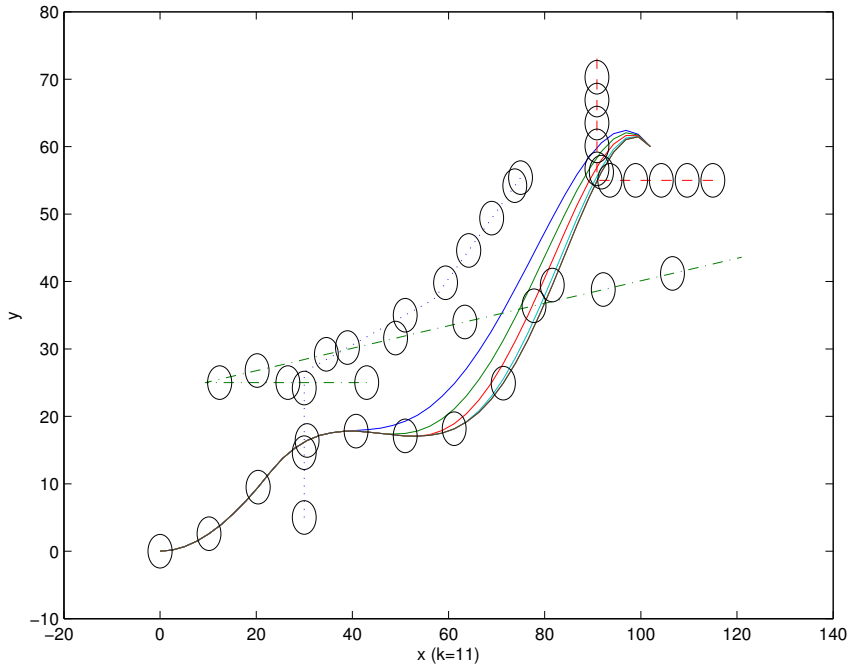

Figure 16: Robot's trajectory for the eleventh sampling instant

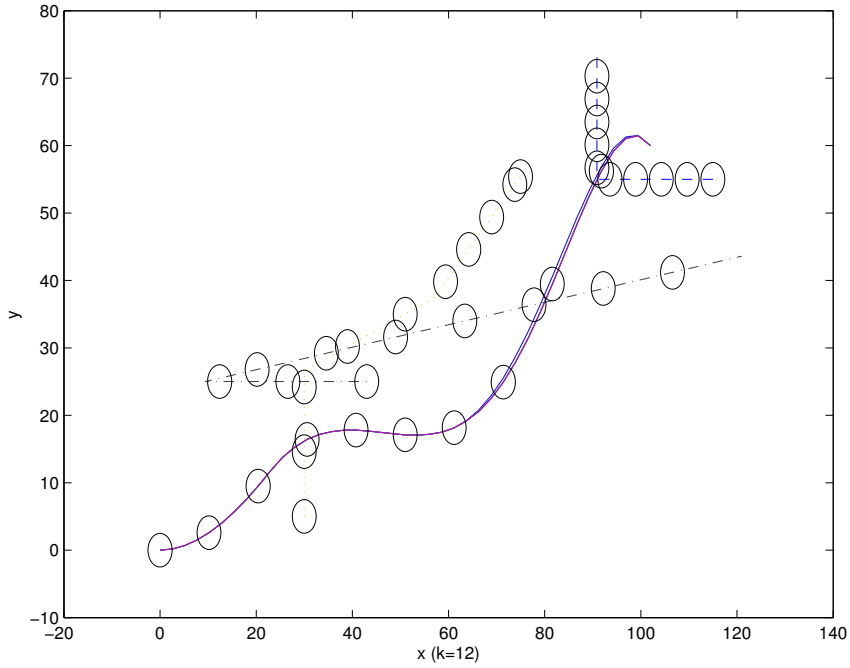

Figure 17: Robot's trajectory for the last sampling instant

The control inputs that drive the robot from specified initial location to desired location collision free are shown below.

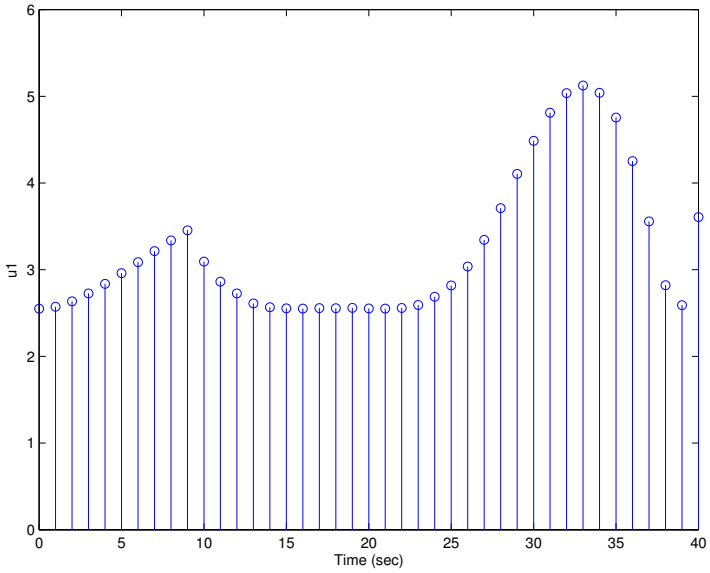

Figure 18: Speed Control (inch/second)

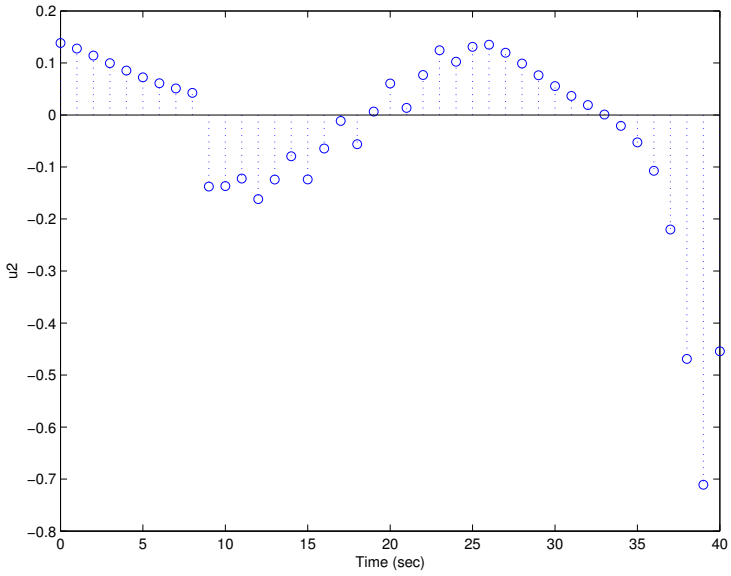

Figure 19: Steering Control (inch/second)

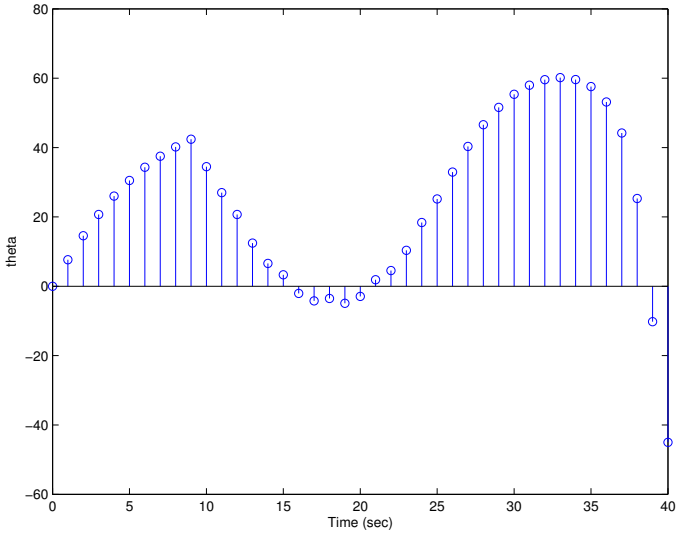

Figure 20: Angle in degree

In table 1, the 1 or 0 value under obstacle columns represent that obstacle is being seen by the robot or not being seen by the robot during time interval k.

Table 1: A Snapshot of Obstacles Detection and the Resulting a4

k	*Object1(k)*	*Object2(k)*	*Object3(k)*	*a4(k)* $*10^4$
1	1	0	0	0
2	1	1	0	0
3	1	1	0	0
4	1	1	0	−0.0996
5	1	1	0	−0.1282
6	1	1	0	−0.1708
7	1	1	0	−0.2276
8	1	1	0	−0.3046
9	1	1	0	−0.3685
10	1	1	0	−0.3736
11	1	1	1	−0.3736
12	1	1	1	−0.3736

4 IMPLEMENTATION

4.1 Introduction

Global positioning system (GPS) allows a relatively precise dynamic measurement in the open considering how big the Earth is. Using a combination of GPS, inertial sensor, compass, and encoder, a desired precision can be obtained. Unfortunately, GPS does not work inside a building. So dynamic measurement has to be done based on dead-reckoning technique that depend on the assumption that the robot know where it begin. As the wheel roll, encoder records the distance and direction travel. At the same time a compass and inertial sensor do their part by measuring the bearing and the rate of turn respectively. Combining all values obtained by encoder, compass, and inertial sensor, a precise measurement allow accurate determination and better control of the robot's movement. However, under careful experimentation, an exceptional result can be obtained without the use of inertial sensor, especially when the robot is very slow.

To communicate between the pc and a robot, a wireless system based on frequency modulation is used. A specific pc interface is written in C++ to communicate with the robots and all of the obstacles. The robot's and obstacles' code are written in Visual Basic. The way pc communicate with the robot and obstacles will be discussed in section 4.4.

4.2 Robot Hardware

Every robot and obstacle contains a microcontroller (OOPIC-R), a fast wireless communication module (FWCM), a magnetic compass, and an encoder.

They will be described in the following order: microcontroller, FWCM, encoder set, and dc motors.

Microcontroller

The OOPic-R board includes a serial port, 16 bidirection lines, serial LCDs, I2C network, two voltage regulators, three programmable push buttons, three indicator LED's, a speaker, and the OOPic2+ firmware built on the capabilities of the PIC16F877. Below is a picture of an OOPic-R microcontroller:

Figure 21: an OOPic-R

The BrainStem Moto 1.0 Module offers two channels of high resolution motion control. The two channel offer flexible pulse width modulation or proportional integral derivative control of motors with various types of feedback including encoders, quadrature encoders, analog input, and Back-EMF speed control.

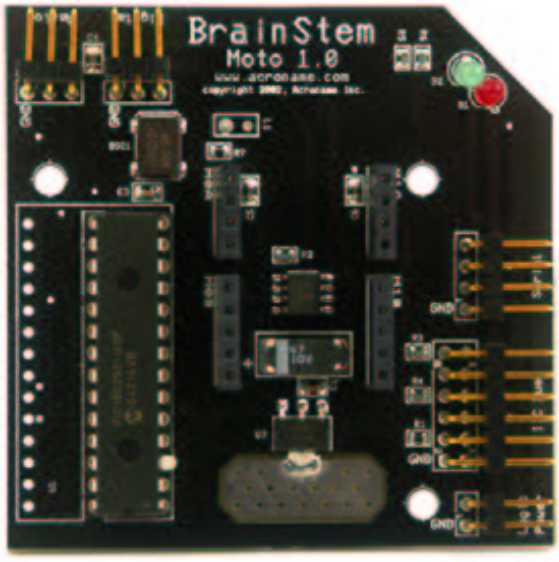

Figure 22: BrainStem Moto 1.0 Board

Wireless Communication Module

Fast Wireless Communication Module or FWCM can communicate with up to 255 other FWCMs as far away as 300 meters. FWCM can transmit or receive 9 bytes at a time in just 200 milliseconds. Having 100 percent data accuracy with packet checking and error correction and anti-collision, FWCM is an ideal choice for this experimentation. There are 3 available modes in FWCM such as normal, broadcast, and autonomous. Any mode can be used at any time, but only normal and broadcast mode allow a network of up to 255 nodes. In autonomous mode, no more than 4 nodes network is allowed. Normal mode allows bidirectional transmission among nodes, but only two nodes at a time. Broadcast mode allows a node to communicate up to 254 other nodes at one time. The protocal to communicate with other FWCM will be discussed in section 4.4.

Figure 23: FWCM

Encoder

Encoder below is specifically for standard servo motors which have been modified to dc motors. It produces standard channel A/channel B raw quadrature outputs, decoded clock, and direction signals. With just channel A/channel B outputs, speed and distance travel can be measured. In each rotation of a wheel, 128 clock ticks is counted. Since the wheel of a robot and obstacles have a diameter of approximately 2.63 inches, the circumference of each wheel is 8.23 inches. Therefore, 15.5 clock ticks is equal to an inch.

Figure 24: Encoder Set

DC Motor

The dynamic of each motor is very similar to each other. The dc motors used in robot 1, robot 2, robot 3, and robot 4 have characteristics as illustrated by the following figures:

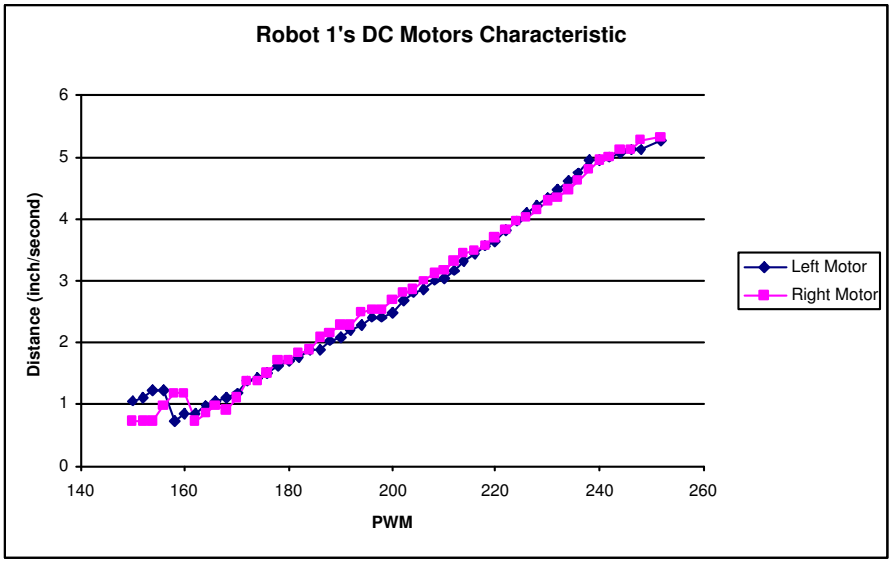

Figure 25: Obstacle 1's DC Motors Characteristic

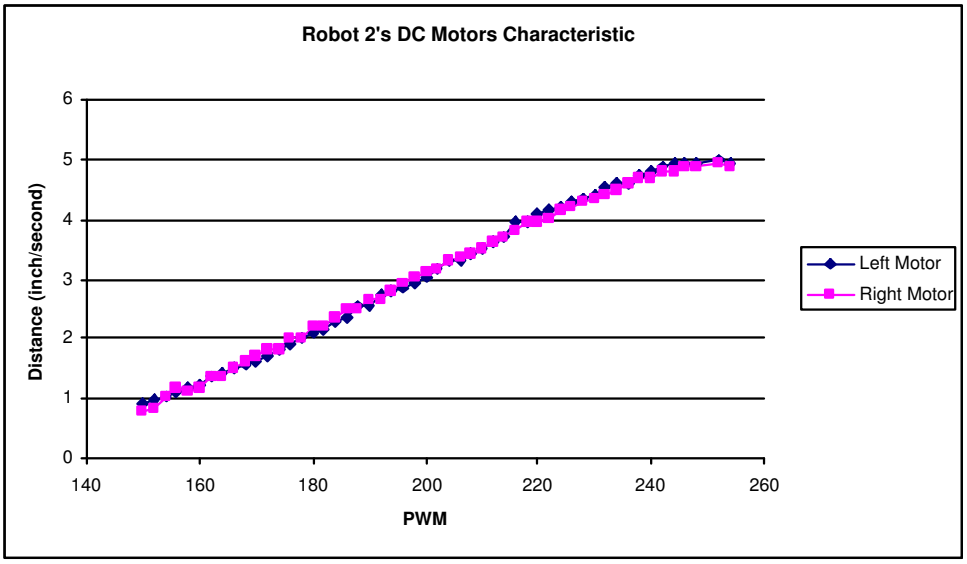

Figure 26: Obstacle 2's DC Motors Characteristic

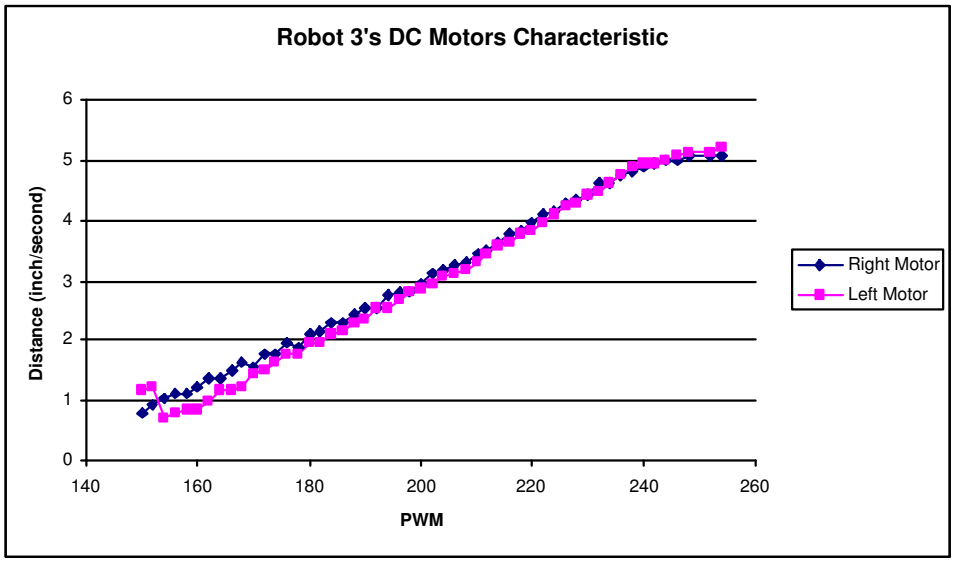

Figure 27: Obstacle 3's DC Motors Characteristic

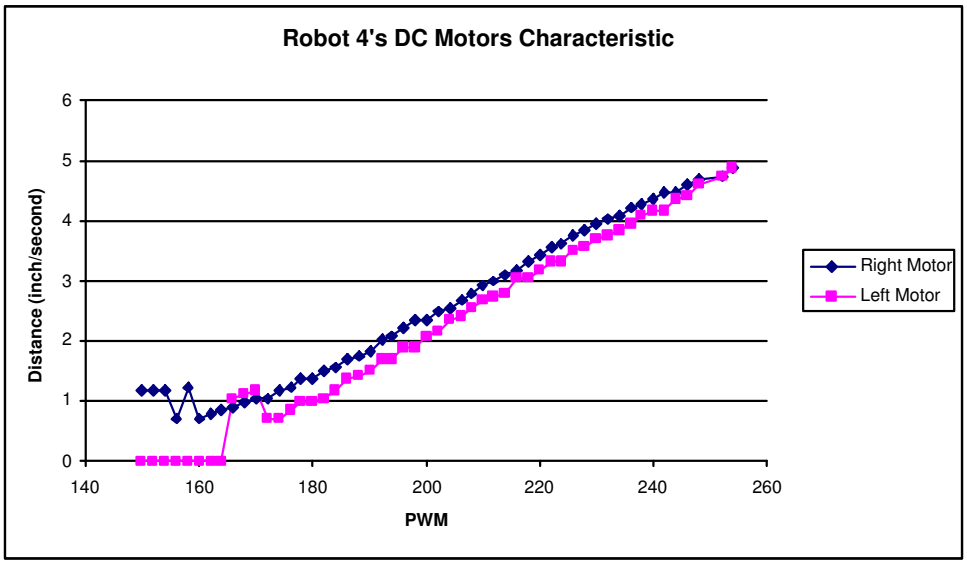

Figure 28: Main Robot's DC Motors Characteristic

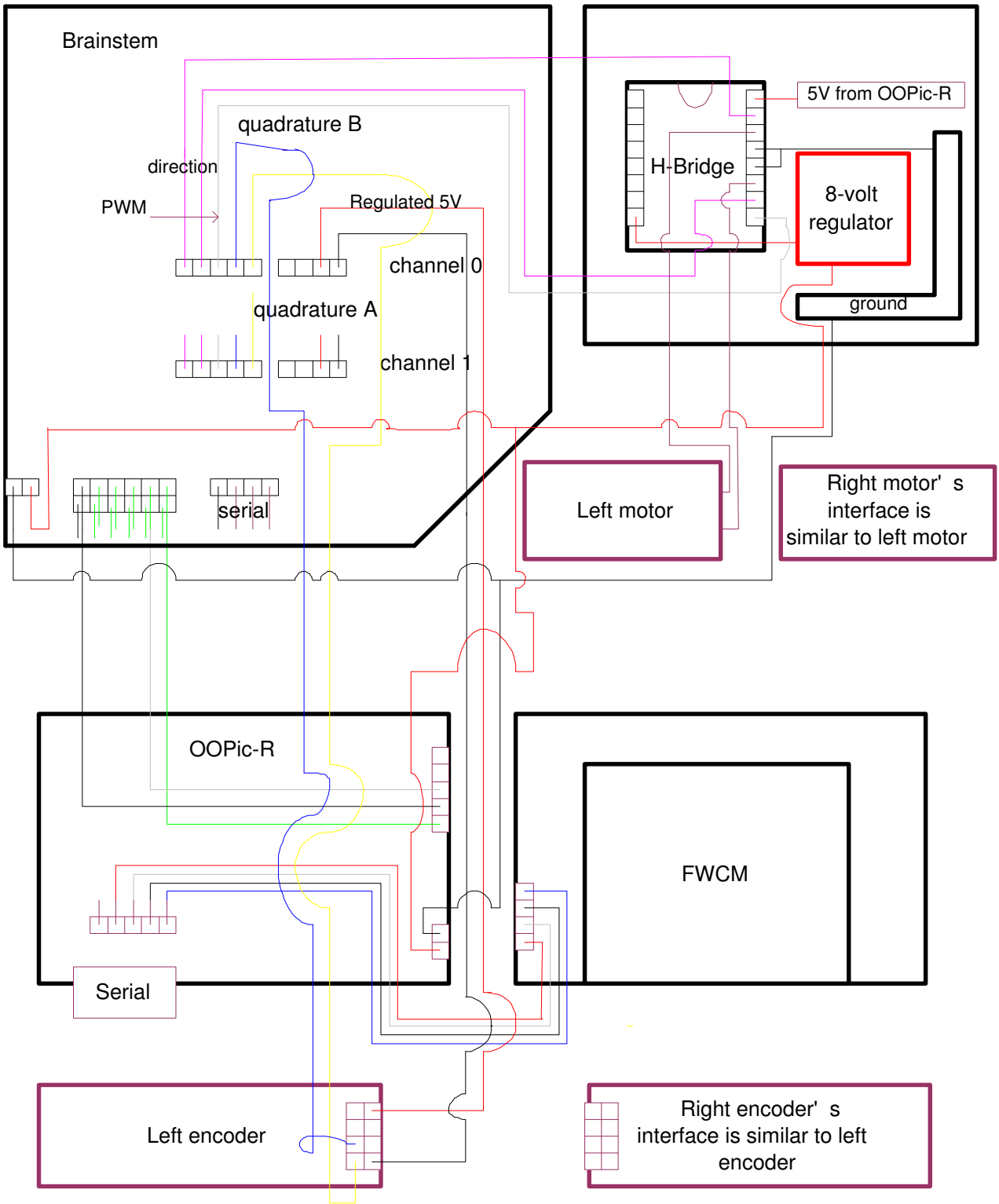

Figure 29: Main Robot's Hardware Interface

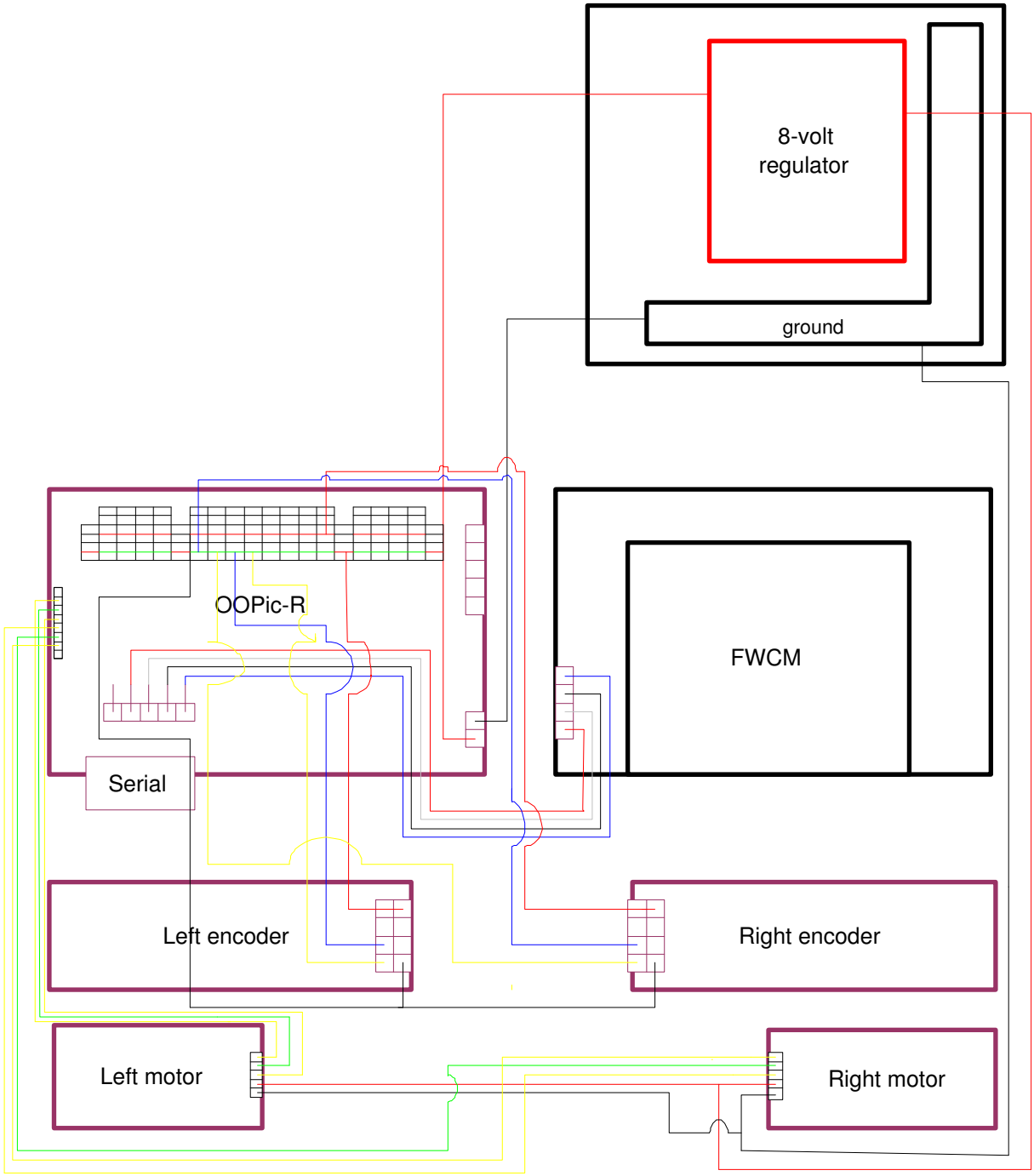

Figure 30: Obstacles' Hardware Interface

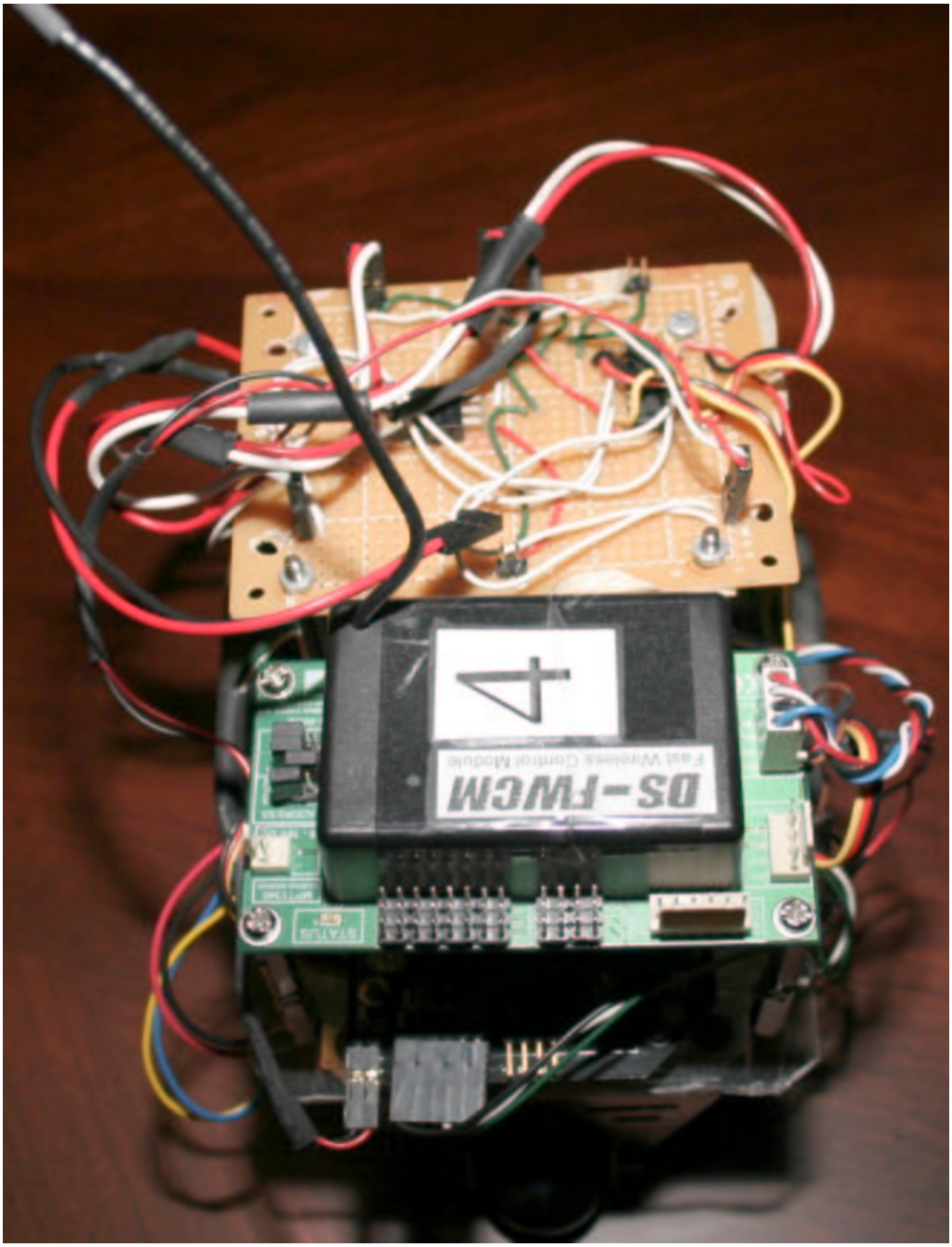

Figure 31: Main Robot

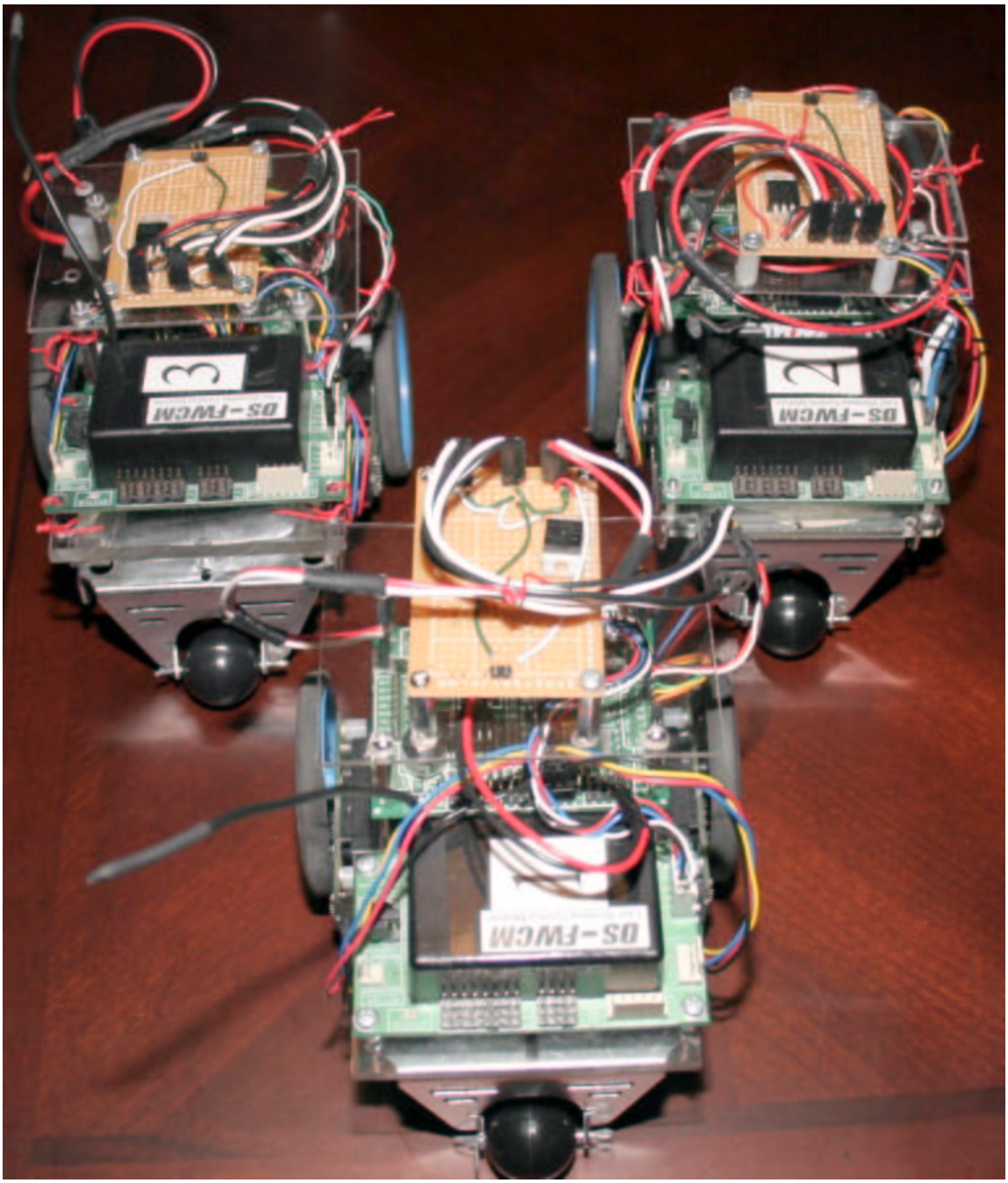

Figure 32: The Three Obstacles

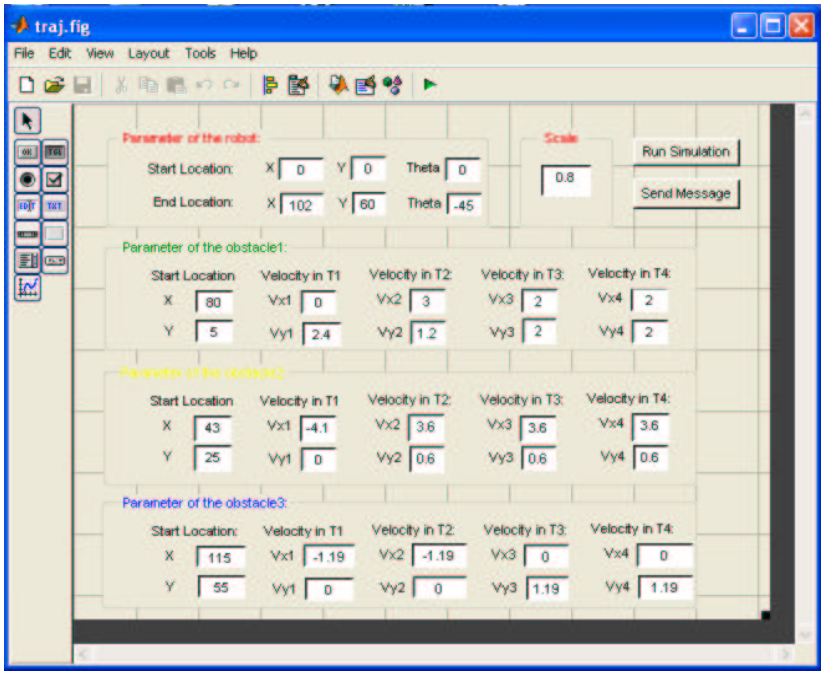

Figure 33: PC-FWCM Interface

4.3 PC Interface

There are four different PC interfaces for this application. The first is a specific pc interface between the pc and FWCM I wrote for this experimentation. The second one is the oopic interface which is provided by www.oopic.com. It is used to write basic program and download into the $oopic-r$'s eeprom. The third one is the moto interface provided by www.acroname.com to tune the pid velocity control of both wheels. The last one is the console interface provided by www.acroname.com to program the Brainstem moto 1.0 module and download the code into the board. Below are pictures of all of the interfaces mentioned above:

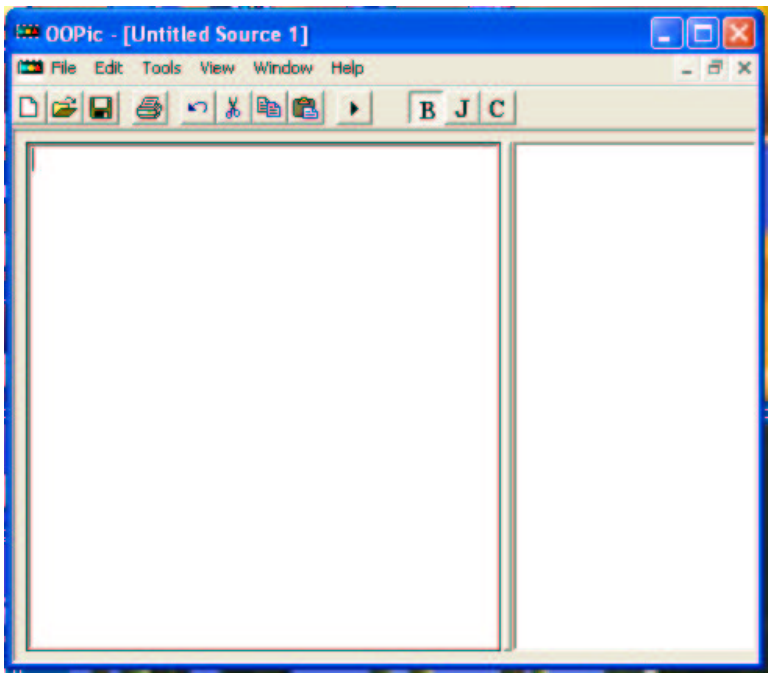

Figure 34: PC-OOPic R Interface

Figure 35: PC-PID Interface

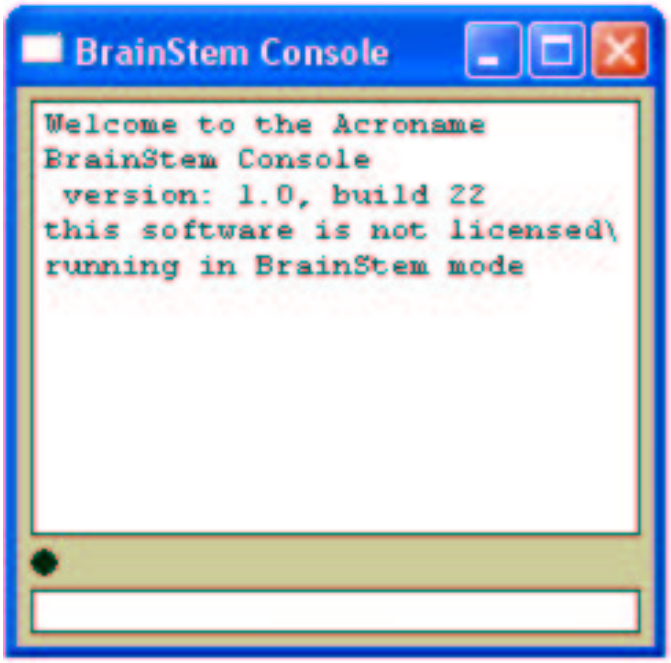

Figure 36: PC-Brainstem Interface

4.4 Transmitting And Receiving Protocals

Figure 33 illustrates a methodological controlling structure from a pc to the robot. Figure 34 illustrates the bigger picture of how a pc communicate with the robot and moving obstacles. In this experimentation, there are only 3 moving obstacles. However, the number of moving obstacles can be expanded to 253 if there are only one robot and only one computer controlling them. Follow are top down approaches for the PC to communicate with a robot and obstacles.

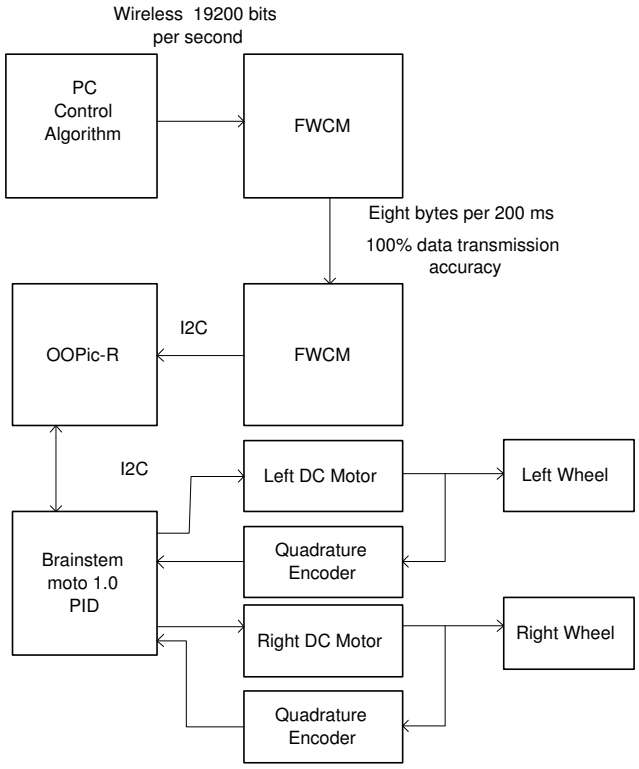

Figure 37: PC to Robot control architecture

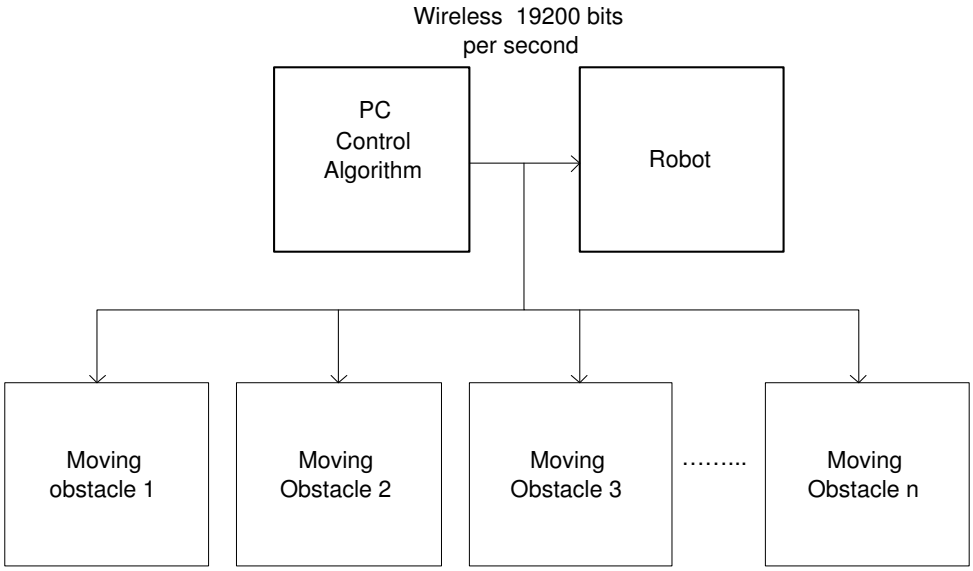

Figure 38: PC to Robot and Obstacles control architecture

PC-FWCM Side

A protocal to send data from a PC to the robot and other obstacles is very simple. First a pc, a robot, and obstacles must have node addresses that are within range of 1 and 255. Their node address cannot be the same if they are to be unique. However, their node addresses can be the same if they are used in broadcast mode and if they are using the same data. To send data from a PC to FWCM through serial port, the following must be pre-configure:

Comport = any,

Rate = 19200,

Stopbits = 1,

Parity = No.

FWCM can send and receive up to 9 bytes in 200 ms. To send data, two steps must be followed. First set up a protocal to send one byte or more from

a pc side. In C++Builder, a buffer to hold integers to be send to remote node must be declare as *unsigned char send[18]*,

send[0] = 91,

send[1] = 192,

send[2] = 0,

send[3] = 7,

send[4] = 254,

send[5] = 3,

send[6] = byte1,

send[7] = 7,

send[8] = byte2,

send[9] = byte3,

send[10]= byte4,

send[11]= byte5,

send[12]= byte6,

send[13]= byte7,

send[14]= byte8,

send[15]= byte9,

send[16]= 93,

send[17]= 93.

The second step is to send the buffer by issue the command *ComPort− >Write(send,18)*. send[0] contains 91 which is the start bit. send[1] contains 192 which is the PC's FWCM address. send[2] contains 0 which is the register to configure the node address of the PC's FWCM stored in send[3].

The node address of the PC is 7 which is in send[3]. send[4] contains the remote node address of the robot or obstacles, the value in send[4] can be any integer from 1 to 255. send[5] contains timeout value in millisecond. send[6], send[8], send[9], up to send[15] are the bytes to be received by remote robot or remote obstacles. send[7] points to the 7^{th} register. When the 7^{th} register is pointed at, the 8^{th} up to the 15^{th} register will be available to store data. If the 8^{th} register up to the 15^{th} register contain valid value, a valid value is a value in the range from 1 to 255, then the 2^{nd} up to the 9^{th} byte will be store in send buffer to be send to remote node. If any register from register 8 to register 15 contain invalid data, then nothing will be send to remote node when a write command is issued. If send[7] is not declare, then the send buffer only send the first byte when a write command is issued. send[16] and send[17] contain 93 which is the end sequence. FWCM needs 2 end sequences.

Robot-FWCM Side

If sending data from the robot's node is desired, then the same concept above apply. If the PC is wanting to receive data from the robot, the following concept will also apply. To receive data from the pc's node, several steps must be done. For simplicity, a simple and specific example is written to be analyze. The first line defines i2c object so that the OOPic-R can communicate with FWCM. Line 2 defines local node address of the robot or the obstacle. Line 3 to 11 define data the robot expected to receive from the PC. Line 12 defines FWCM's address. The rest are switch statements. Just like most programming languages, Visual Basic require a main prototype.

```
 1  Dim FWCM      As New oi2c
 2  Dim LocNode   As New oByte
 3  Dim byte1     As New oByte
 4  Dim byte2     As New oByte
 5  Dim byte3     As New oByte
 6  Dim byte4     As New oByte
 7  Dim byte5     As New oByte
 8  Dim byte6     As New oByte
 9  Dim byte7     As New oByte
10  Dim byte8     As New oByte
11  Dim byte9     As New oByte
12  Dim ReadAdr   As New oByte
13  Dim Error     As New oBit
14  Dim OError    As New oBit
15  Dim ReadOK    As New oBit
16  Dim Done      As New oBit
17  Dim TOut      As New oByte
```

```
1 Sub Main()
2 oopic.delay = 200
3 LocNode = 2
4 RemNode = 7
5 Const WriteTOut = 3
6 Const FWCMAdr = &h61
7 Call SetUpFWCM
8 Call Read_Data
9 End Sub
```

It take 2 seconds to initialize the FWCM as seen at line 2 above. Lines 3 and 4 assign local and remote node addresses or robot and the pc's node addresses respectively. Line 5 assigns timeout to be 3 milliseconds. Line 6 assigns FWCM's address. Lines 7 and 8 call other functions. The functions below are very much selves explanatory.

```
Sub Read_Data()
1 Do
2 do
3 call  ReadFWCM
4 loop until ReadOK = cvTrue
5 loop '  keep reading
End Sub
Sub SetUpFWCM()
8       FWCM.Node = FWCMAdr          '  Setup I2C address for FWCM
9       '  Setup I2C addressing to FWCM
10      FWCM.Width = cv8bit          '  Control Info is 1-byte
11      FWCM.Mode  = cv10bit         '  I2C mode is 10-Bit Addressing
12      FWCM.NoInc = cvFalse         '  Increment on every read/write
13      FWCM.Location = 0            '  point to local node address
14      FWCM.Value = LocNode         '  and write local node address
End Sub
```

The function above is the most important function. The purpose of the Read-FWCM is to receive meaningful and useful data. No data can be lost. The received data will be used to drive the robot and obstacles. Line 6 in the function above pointed to register 20 so that up to 9 bytes can be read and store. It is analogous to register 7 for writing up to 9 bytes.

```
Sub ReadFWCM()
 1  FWCM.Location = 2              '  Start at read status flag (R2)
 2  If FWCM.Value = 255 then       '  Has data been received (R2) ?
 3  If FWCM.Value  = 0 then         '  Is there overflow?
 4  ReadAdr  = FWCM.Value          '  If not then store FWCM address
 5   byte1    = FWCM.Value
 6   FWCM.Location = 20
 7   byte2    = FWCM.Value
 8   byte3    = FWCM.Value
 9   byte4    = FWCM.Value
10   byte5    = FWCM.Value
11  byte6    = FWCM.Value
12   byte7    = FWCM.Value
13   byte8    = FWCM.Value
14   byte9    = FWCM.Value
15   OError   = cvFalse           '  Indicate no overflow error
16   ReadOK   = cvTrue    '  Indicate data has been read and stored
17 else
18   OError   = cvTrue              '  Indicate overflow error has occured
19   ReadOK   = cvTrue               '  Indicate data has been read
20   ReadAdr  = FWCM.Value
21 end if
22 else
23  OError = cvFalse               '  If no data read then no error
24  ReadOK = cvFalse               '  and no data
25 end if
End Sub
```

OOPic-R and Brainstem Communication

The $OOPic-R$ and Brainstem communicate with each other through I2C protocal in the same way that $OOPic-R$ communicate with FWCM. Just like the $OOPic-R$'s way of obtaining its data from FWCM, the Brainstem's way of obtaining the data from the $OOPic-R$ is the same way. Follow is an example of how the $OOPic-R$ and the Brainstem communicate with each other.

The figure below is the program that have to be downloaded to the $OOPic-R$.

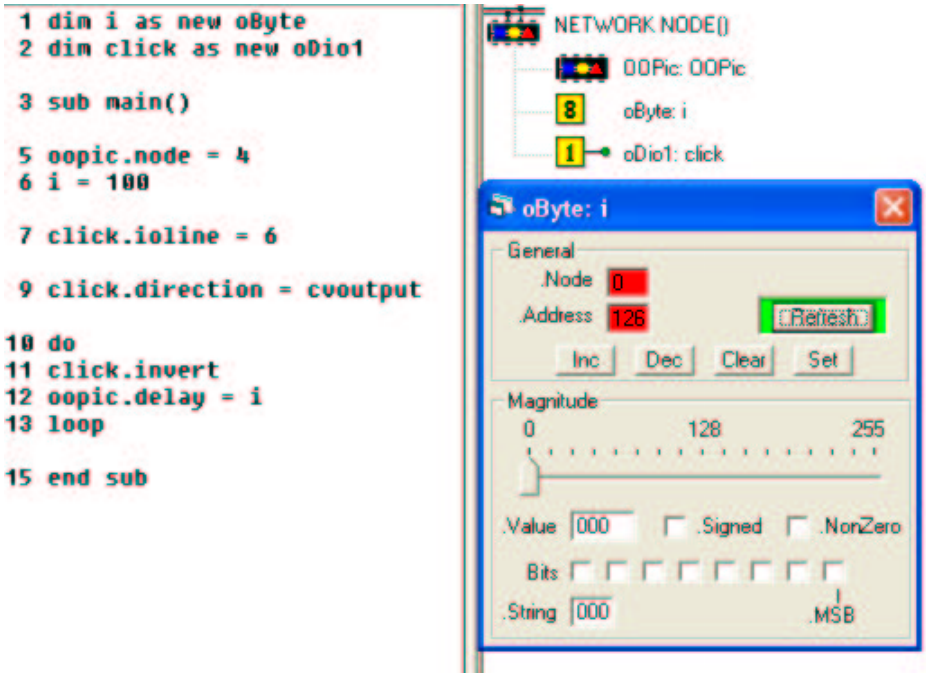

Figure 39: OOPic-R Interface

Line number 5 enable I2C communication with any device, the Brainstem in this case, that have I2C network. The address of the I2C network on the $oopic-R$ is set to 4. The oByte i address which is assigned by the $OOPic-R$ compiler is 126. The figure below is the program that have to be downloaded to the brainstem. All of the functions seen in that figure are

```
1 void main()
2 {
3   char x;

5   aCore_Sleep(30000);

6   while (1)
7   {
8     aOOPic_WriteChar(8,(char)ADDR_I_VALUE,0,2);
9     x=aOOPic_ReadChar(8,(char)ADDR_I_VALUE);
10    aPrint_CharDec(x);
11    aPrint_Char('\n');
12    aCore_Sleep(20000);

14    aOOPic_WriteChar(8,(char)ADDR_I_VALUE,0,20);
15    x=aOOPic_ReadChar(8,(char)ADDR_I_VALUE);
16    aPrint_CharDec(x);
17    aPrint_Char('\n');
18    aCore_Sleep(20000);
19  }
20 }
```

provided by Acroname. The first argument in $aOOPic-WriteChar(ADDR-OOPIC, (char)ADDR-I-VALUE, 0, 2)$ is the address of the $OOPic-R$ as seen by the Brainstem; it is 8 in this example. I2C networking require the address to be shifted left by 1 bit. The purpose of the example illustrated here is to show how the byte i in an $OOPic-R$ can be alter to any 8-bit value by the Brainstem. The first alteration was done when the Brainstem write a value of 2 to the $OOPic-R$. The second time, the Brainstem write a value of 20. Note that the fourth argument in the $aOOPic-WriteChar(ADDR-OOPIC$,(char)$ADDR-I-VALUE$,0,2) is the written values.

4.5 Result

All of the components used to build the robot has some errors associate with them. The robot platform contribute some errors such as slight different in wheels and the plate that support the robot. This is known as a kinematic imperfection of a mobile robot. The error can be reduce by following the

method used in [11]. A major error contributor is the rounding error. The OOPic controller is configure to use an eight-bit integer, although it can be difficultly configure to use as sixteen-bit integer. Since a lot of rounding were made, a noticeable error is seen. Although there are problem from each and every component, the algorithm provide a very good experimentation result in locally closed loop control at the robot's end.

5 CONCLUSION

The algorithm in [1] is indeed a very useful algorithm that provide analytical solution to a class of nonholonomic system. From a simple two-wheel robot, mathematical equations representing the robot's kinematic and dynamic movement were derived, proved, analyzed, simulated, and implemented successfully. The algorithm, although very good, can be improved so that the robot can take less time to avoid the obstacle.

My contributions include designing and building wireless mini robots, writing pc interface in C++ Builder, then matlab, and writing codes to programs the robot and moving obstacles to perform a collision-free trajectory generation. The robot and moving obstacles can easily be reprogram to perform other useful tasks such as formation control. This thesis provided me a lot of experience and insight into theoretical research and practical implementation.

APPENDIX A: CONTROL ALGORITHM CODE IN MATLAB

matlab code

clc;

```
global  Fx0 Fy0 Fxf Fyf Ftheta0 Fthetaf p1x p1y p2x p2y p3x p3y p1vx1 p1vx2 p1vx3 p1vy1
p1vy2 p1vy3
global  p2vx1 p2vx2 p2vx3 p2vy1 p2vy2 p2vy3  p3vx1 p3vx2 p3vx3 p3vy1 p3vy2 p3vy3
global  p1vx4 p1vy4 p2vx4 p2vy4 p3vx4 p3vy4  ratio  VR VL Tmedium
%Sensing Range
obstacle = 1;
Rs = 25;

%The boundary conditions in original space
%q0=(x0,y0,theta0)^T;
x0 = Fx0;
y0 = Fy0;
theta0 = Ftheta0*pi/180;

xf = Fxf;
yf = Fyf;
thetaf = Fthetaf*pi/180;

%The boundary conditions in transformed space
%initial point: z0
[z10, z20, z30] = ChainedTransform(x0, y0, theta0);

%final point: zf
[z1f, z2f, z3f] = ChainedTransform(xf, yf, thetaf);

%Time to steer the robot from initial point to final point
T = 40;

%Robot speed
C = (z1f - z10)/T;

%The vector of sampling time instant
Ts = [0 3 6 9 12 15 18 21 24 27 30 33 40];

%We observe the velocity change of objects every 3 seconds.

vo1x = [p1vx1   p1vx1 p1vx1    p1vx2 p1vx2 p1vx2 p1vx3 p1vx3 p1vx3 p1vx4 p1vx4 0*ratio];
vo1y = [p1vy1 p1vy1 p1vy1 p1vy2 p1vy2 p1vy2 p1vy3 p1vy3 p1vy3 p1vy4 p1vy4 0*ratio ];

%Robot4
```

```
%-3*ratio
vo2x = [p2vx1   p2vx1 -3*ratio   p2vx2 p2vx2 p2vx2 p2vx3 p2vx3 p2vx3 p2vx4 p2vx4 p2vx4 ];
vo2y = [p2vy1 p2vy1 p2vy1 p2vy2 p2vy2 p2vy2 p2vy3 p2vy3 p2vy3 p2vy4 p2vy4 p2vy4 ];

%Robot6
vo3x = [p3vx1   p3vx1 p3vx1    p3vx2 p3vx2 p3vx2 p3vx3 p3vx3 p3vx3 p3vx4 p3vx4 p3vx4 ];
vo3y = [p3vy1 p3vy1 p3vy1 p3vy2 p3vy2 p3vy2 p3vy3 p3vy3 p3vy3 p3vy4 p3vy4 p3vy4 ];

%Computing step time
T1 = 1;%1;
tt = 0:T1:T;

%radius of obstacles
r = 5;
%radius of the robot
R = 5;

%initial position of moving obstacles
%Robot 5
xo10 = p1x;
yo10 = p1y;

%obstacle 6
xo20 = p2x;
yo20 = p2y;

%Robot 2
xo30 = p3x;
yo30 = p3y;

%Path of obstacles
xo1(1) = xo10;
yo1(1) = yo10;

xo2(1) = xo20;
yo2(1) = yo20;

xo3(1) = xo30;
yo3(1) = yo30;

%initial states
z1(1) = z10;
z2(1) = z20;
z3(1) = z30;
```

```
x(1) = x0;
y(1) = y0;
theta(1) = theta0;

%initialize a4 of all 3 obstacles
a41max = 0;
a41min = 0;

a42max = 0;
a42min = 0;

a43max = 0;
a43min = 0;

a4max = 0;
a4min = 0;

clc;
c = fix(clock);
fprintf('started at %2i.%2i.%2i\n\n', ...
  c(4),c(5),c(6));

%checking for each time interval

for k=1:(length(Ts)-1)
   a4min = 0;
   a4max = 0;
   acommon1(k) = 0;  %set to 1 if obstacle 1 is detected during time k
   acommon2(k) = 0;  %set to 1 if obstacle 2 is detected during time k
   acommon3(k) = 0;  %set to 1 if obstacle 3 is detected during time k
   %index of changing initial condition
   index = Ts(k)/T1+1;

   [B, Y, A] = CalculateBYA(index, z1, z1f, z3, z3f, z2, z2f);

%1111111111111111111111111111111111111111111111111111111111111111111111111111

 if (obstacle == 0)
    a4(k) = 0;
 else

   if ( ( z3(index)- yo1(index) )^2 +  ( z1(index) - xo1(index) )^2 ) <= (Rs-r)*(Rs-r)
      [tmin1, tmax1] = TimeCheck(xo1, z10, k, index, vo1x, Ts, C, R, r, T);

     if ((tmin1 >=Ts(k)) || (tmax1 <=T))
```

```
      acommon1(k) = 1;  % a switch statement used to check the number of obstacle being seen at
anytime interval
      [a41min, a41max] = DetermineA4(tmin1, tmax1, z10, C, B, A, Y, k, index, xo1, yo1,
a41min, a41max, Ts, vo1x, vo1y, r, R);

     else acommon1(k) = 0;
       a41min = 0;
       a41max = 0;
     end
  else acommon1(k) = 0;
       a41min = 0;
       a41max = 0;
  end

%22222222222222222222222222222222222222222222222222222222222222222222222222

  if ( ( z3(index)- yo2(index) )^2 +  ( z1(index) - xo2(index) )^2 ) <= (Rs-r)*(Rs-r)
      [tmin2, tmax2] = TimeCheck(xo2, z10, k, index, vo2x, Ts, C, R, r, T);

       if ((tmin2 >=Ts(k)) || (tmax2 <=T))
            acommon2(k) = 1;  % a switch statement used to check the number of obstacle if any
being detected at anytime interval
      [a42min, a42max] = DetermineA4(tmin2, tmax2, z10, C, B, A, Y, k, index, xo2, yo2,
a42min, a42max, Ts, vo2x, vo2y, r, R);
       else acommon2(k) = 0;
          a42min = 0;
          a42max = 0;
       end

  else acommon2(k) = 0;
       a42min = 0;
       a42max = 0;
  end

%33333333333333333333333333333333333333333333333333333333333333333333333333
  if ( ( z3(index)- yo3(index) )^2 +  ( z1(index) - xo3(index) )^2 ) <= (Rs-r)*(Rs-r)
       [tmin3, tmax3] = TimeCheck(xo3, z10, k, index, vo3x, Ts, C, R, r, T);

       if ((tmin3 >=Ts(k)) || (tmax3 <=T))
           acommon3(k) = 1;  % a switch statement used to check the number of obstacle being
seen at anytime interval
       [a43min, a43max] = DetermineA4(tmin3, tmax3, z10, C, B, A, Y, k, index, xo3, yo3,
a43min, a43max, Ts, vo3x, vo3y, r, R);
       else acommon3(k) = 0;
         a43min = 0;
```

```
            a43max = 0;
        end
    else
          acommon3(k) = 0;
            a43min = 0;
            a43max = 0;
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%check the number of obstacle during each time interval and calculate
%a4(k), obstacle avoidance variable, accordingly.

 if ( (acommon1(k) == 0) && (acommon2(k) == 0) && (acommon3(k) == 0) ) % if no obstacle
is detected at k interval
    a4(k) = 0; %no need to avoid obstacle
 end
 if ( (acommon1(k) == 0) && (acommon2(k) == 0) && (acommon3(k) == 1) )  %if only
obstacle 3 is detected
    %find a4(k) based on second order nonlinear inequality
    a4(k) = DetermineA4k(a43min, a43max);
 end
 if ( (acommon1(k) == 0) && (acommon2(k) == 1) && (acommon3(k) == 0) )  %if only
obstacle 2 is detected
    %find a4(k) based on second order nonlinear inequality
    a4(k) = DetermineA4k(a42min, a42max);
 end
 if ( (acommon1(k) == 0) && (acommon2(k) == 1) && (acommon3(k) == 1) )  %if only 2
obstacles, 2 and 3, are detected
    %find a4(k) based on 2 second order nonlinear inequalities
    a4(k) = DetermineA4k2(a42min, a42max, a43min, a43max, a4min, a4max);
 end
 if ( (acommon1(k) == 1) && (acommon2(k) == 0) && (acommon3(k) == 0) )  %if only
obstacle 1 is detected
    %find a4(k) based on 2 second order nonlinear inequalities
    a4(k) = DetermineA4k(a41min, a41max);
 end
 if ( (acommon1(k) == 1) && (acommon2(k) == 0) && (acommon3(k) == 1) )  %if only 2
obstacles, 1 and 3, are detected
    %find a4(k) based on 2 second order nonlinear inequalities
     a4(k) = DetermineA4k2(a41min, a41max, a43min, a43max, a4min, a4max);
  end
  if ( (acommon1(k) == 1) && (acommon2(k) == 1) && (acommon3(k) == 0) )  %if only 2
obstacles, 1 and 2, are detected
    %find a4(k) based on 2 second order nonlinear inequalities
```

```
      a4(k) = DetermineA4k2(a41min, a41max, a42min, a42max, a4min, a4max);
  end
  if ( (acommon1(k) == 1) && (acommon2(k) == 1) && (acommon3(k) == 1) )  %if all 3
obstacles, 1, 2, and 3, are detected
     %find a4(k) based on 3 second order nonlinear inequalities
     a4(k) = DetermineA4k3(a41min, a41max, a42min, a42max, a43min, a43max, a4min, a4max)
 end

end %end if there is obstacle or no obstacle

% calculate the remaining coefficients a0 to a3
    a0123=inv(B)*Y-inv(B)*A*a4(k); % 4x1 matrix
    a0(k)=a0123(1);
    a1(k)=a0123(2);
    a2(k)=a0123(3);
    a3(k)=a0123(4);

% calculate the steering inputs:
    C0(k)=2*a2(k)*C+ 6*a3(k)*z1(index)*C + 12*a4(k)*z1(index)^2*C;
    C1(k)=6*a3(k)*C^2 + 24*a4(k)*z1(index)*C^2;
    C2(k)=12*a4(k)*C^3;

% Produce the trajectory history of robot and obstacles

    for j=index:length(tt)
    % the trajectory in z plane

        z1(j) = z10+C*tt(j);
        z2(j) = z2(index)+C0(k)*(tt(j)-tt(index))+C1(k)*(tt(j)-tt(index))^2/2+...
            C2(k)*(tt(j)-tt(index))^3/3 ;
        z3(j) = z3(index)+C*z2(index)*(tt(j)-tt(index))+C*C0(k)*(tt(j)-tt(index))^2/2+...
            C*C1(k)*(tt(j)-tt(index))^3/6+C*C2(k)*(tt(j)-tt(index))^4/12;

    % the trajectory in x-y plane
        theta(j)=atan(z2(j));
        Tita(j) = theta(j)*180/pi;
        x(j)=z1(j);
        y(j)=z3(j);

    % steering input
        v1(j)=C;
        v2(j)=C0(k)+C1(k)*(tt(j)-tt(index))+C2(k)*(tt(j)-tt(index))^2;

    % real input
        u1(j)=v1(j)*sec(theta(j));
```

```
      u2(j)=v2(j)*(cos(theta(j)))^2;

  % the motion of obstacles:
      xo1(j)=xo1(index)+vo1x(k)*(tt(j)-tt(index));  % The current position of obstacle 1;
      yo1(j)=yo1(index)+vo1y(k)*(tt(j)-tt(index));

      xo2(j)=xo2(index)+vo2x(k)*(tt(j)-tt(index));  % The current position of obstacle 2;
      yo2(j)=yo2(index)+vo2y(k)*(tt(j)-tt(index));

      xo3(j)=xo3(index)+vo3x(k)*(tt(j)-tt(index));  % The current position of obstacle 3;
      yo3(j)=yo3(index)+vo3y(k)*(tt(j)-tt(index));
   end

DTita(1) = atan2((y(2)-y(1)),(x(2)-x(1)))-0;

for i=1:(length(tt) - 2)
  angel1(i)=atan2((y(i+2)-y(i+1)),(x(i+2)-x(i+1)))*180/pi;
  angel2(i)=atan2((y(i+1)-y(i)),(x(i+1)-x(i)))*180/pi;
  DTita(i+1) = (angel1(i)-angel2(i));
end

for i=1:(length(tt) - 1)

   Tmedium(i) = 1;

  if (u2(i) > 0)

    %speed in term of inch/second
    VR(i)  = ((2*u1(i)+u2(i)*4.74)/2); %Turn is Right Wheel
    VL(i)  = (2*u1(i) - VR(i));      %Tturn is Left Wheel

    %speed in term of setpoint
    VR(i) = round(VR(i)*(128/(8.23))); %128 is the number of clock ticks per wheel revolution
    VL(i) = round(VL(i)*(128/(8.23))); %8.23 is the 2*pi*r (r=1.31 inches)
    %wireless data exchange only work with integer and with number
    %greater than 0


    if (VR(i) < 0)
       VR(i) = 0;
    end

    if (VL(i) < 0)
       VL(i) = 0;
    end
```

```
end


if (u2(i) < 0)

    %speed in term of inch/second
    VL(i)  = (0.5*( abs(u2(i))*4.74 + 2*u1(i) ));
    VR(i)  = (2*u1(i) - VL(i));

    %speed in term of setpoing
   VR(i) = round(VR(i)*(128/(8.23))); %128 is the number of clock ticks per wheel revolution
   VL(i) = round(VL(i)*(128/(8.23))); %8.23 is from 2*pi*r (r=1.31 inches)

   if (VR(i) < 0)
      VR(i) = 0;
   end

   if (VL(i) < 0)
      VL(i) = 0;
   end

end


    VR(i) = VR(i) + 1;
    VL(i) = VL(i) + 1;
 end

 for p=1:40
 VL(p)
 end

 % keep the comparision trajectories
 if k==1
     x1=x;
     y1=y;
 end
 if k==2
     x2=x;
     y2=y;
 end
 if k==3
```

```
        x3=x;
        y3=y;
    end
    if k==4
        x4=x;
        y4=y;
    end
    if k==5
        x5=x;
        y5=y;
    end
    if k==6
        x6=x;
        y6=y;
    end
    if k==7
        x7=x;
        y7=y;
    end
    if k==8
        x8=x;
        y8=y;
    end
    if k==9
        x9=x;
        y9=y;
    end
    if k==10
        x10=x;
        y10=y;
    end
    if k==11
        x11=x;
        y11=y;
    end
    if k==12
        x12=x;
        y12=y;
    end

end %k=1:(length(Ts)-1)
Limit = T/4;
c=fix(clock);
fprintf('end at %2i.%2i.%2i\n\n', ...
   c(4),c(5),c(6));
```

```
for j=1:length(tt)  % length(tt) = 401
  ttt(j)=tt(j);
end

R=2*R;
r=2*r;
figure
plot(x1,y1,'b-', xo1, yo1,'g:', xo2, yo2, 'r-.', xo3, yo3,'c--');
hold on
for i=1:Limit

   index1=4/T1*(i-1)+1; % every 3 sec, draw position of robot and objects
   rectangle('Curvature',[1 1],'EdgeColor','b', 'Position', [x1(index1)-R/2 y1(index1)-R/2 R R]);

   %if (obstacle  ~= 0)
   rectangle('Curvature',[1 1], 'EdgeColor','g', 'Position', [xo1(index1)-r/2 yo1(index1)-r/2 r r]);
   rectangle('Curvature',[1 1], 'EdgeColor','r', 'Position', [xo2(index1)-r/2 yo2(index1)-r/2 r r]);
   rectangle('Curvature',[1 1], 'EdgeColor','c', 'Position', [xo3(index1)-r/2 yo3(index1)-r/2 r r]);
   %end

end
hold off

xlabel('x'); % (k=1)
ylabel('y');

figure
plot(x4,y4,'b-', xo1, yo1,'g:', xo2, yo2, 'r-.', xo3, yo3,'c--');
%x1,y1,'-', x2, y2, '-', x3,y3, '-',

hold on
for i=1:Limit

   index1=4/T1*(i-1)+1; % every 3 sec, draw position of robot and object

   rectangle('Curvature',[1 1], 'EdgeColor','b','Position', [x4(index1)-R/2 y4(index1)-R/2 R R]);
   %if (obstacle  ~= 0)
   rectangle('Curvature',[1 1], 'EdgeColor','g', 'Position', [xo1(index1)-r/2 yo1(index1)-r/2 r r]);
   rectangle('Curvature',[1 1], 'EdgeColor','r', 'Position', [xo2(index1)-r/2 yo2(index1)-r/2 r r]);
   rectangle('Curvature',[1 1], 'EdgeColor','c', 'Position', [xo3(index1)-r/2 yo3(index1)-r/2 r r]);
   %   end
end
hold off
xlabel('x');% (k=4)
```

```
ylabel('y');

figure
plot(x12,y12,'b-', xo1, yo1,'g:', xo2, yo2, 'r-.', xo3, yo3,'c--');
%x8,y8,'-', x9,y9,'-', x10,y10,'-', x11,y11,'-',
hold on
for i=1:Limit

   index1=4/T1*(i-1)+1; % every 3 sec, draw position of robot and object

   rectangle('Curvature',[1 1], 'EdgeColor','b','Position', [x12(index1)-R/2 y12(index1)-R/2 R R]);
   %if (obstacle  ~= 0)
   rectangle('Curvature',[1 1], 'EdgeColor','g', 'Position', [xo1(index1)-r/2 yo1(index1)-r/2 r r]);
   rectangle('Curvature',[1 1], 'EdgeColor','r', 'Position', [xo2(index1)-r/2 yo2(index1)-r/2 r r]);
   rectangle('Curvature',[1 1], 'EdgeColor','c', 'Position', [xo3(index1)-r/2 yo3(index1)-r/2 r r]);
   %   end
end
hold off
xlabel('x'); % (k=12)
ylabel('y');

figure
stem(ttt,theta*180/pi);
ylabel('theta');
xlabel('Time (sec)');

figure
stem(ttt, u2, ':');
ylabel('u2');
xlabel('Time (sec)');

figure
stem(ttt,u1,'-');
ylabel('u1');
xlabel('Time (sec)');

function [B, Y, A] = CalculateBYA(index, z1, z1f, z3, z3f, z2, z2f);
%As its name implied, this function is used to calculate matrices: B,Y,A

   B=[ 1 z1(index) (z1(index))^2   (z1(index))^3;
      0    1     2*z1(index)   3*(z1(index))^2;
      1   z1f       z1f^2          z1f^3;
      0    1        2*z1f          3*z1f^2];

   Y=[z3(index); z2(index); z3f; z2f];
```

```
   A=[(z1(index))^4; 4*(z1(index))^3; z1f^4; 4*(z1f)^3];

function [z1, z2, z3] = ChainedTransform(x, y, theta);
%This function is used to transform the given space into chained form space
 z1 = x;
 z2 = tan(theta);
 z3 = y;

function [a4min, a4max, g0] = DetermineA4(tmin, tmax, z10, C, B, A, Y, k, index, xo, yo,
a4min, a4max, Ts, vox, voy, r, R);
%As its name implied, this function is used to determine a4min and a4max for all
%obstacles with the many given input parameters

   tau=tmin:0.1:tmax;

   for i=1:length(tau)
      z(i)=z10+C*tau(i);
      g2=(z(i)^4-[1 z(i) z(i)^2 z(i)^3]*inv(B)*A)^2;
      g1=2*(z(i)^4-[1 z(i) z(i)^2 z(i)^3]*inv(B)*A)*([1 z(i) z(i)^2 z(i)^3]*inv(B)*Y-
voy(k)*(tau(i)-Ts(k))-yo(index));
      g0=([1 z(i) z(i)^2 z(i)^3]*inv(B)*Y-voy(k)*(tau(i)-Ts(k))-yo(index))^2+(z(i)-xo(index)-
vox(k)*(tau(i)-Ts(k)))^2-(r+R)^2;
      b4ac=g1^2-4*g2*g0;

     if (g2 ~= 0)
      if b4ac>=0
      if ( sign((-g1-sqrt(b4ac))/(2*g2)) ~= sign((-g1+sqrt(b4ac))/(2*g2)) )
           if (-g1-sqrt(b4ac))/(2*g2)<a4min
              a4min=(-g1-sqrt(b4ac))/(2*g2);
           end
           if (-g1+sqrt(b4ac))/(2*g2)>a4max
              a4max=(-g1+sqrt(b4ac))/(2*g2);
           end
      end
      end
     end

     if ( (g2 == 0) || (b4ac < 0) )
      a4min = 0;
      a4max = 0;
     end

   end
```

```
function a4k = DetermineA4k2(a4imin, a4imax, a4jmin, a4jmax, a4min, a4max);

if min([a4imin, a4jmin]) < a4min
   a4min = min([a4imin, a4jmin]);
end
if max([a4imax, a4jmax]) > a4max
   a4max = max([a4imax, a4jmax]);
end
if ( sign(a4min) == sign(a4max)  )
   a4k = 0;
else
   if abs(a4min) <= abs(a4max)
      a4k = a4min;
   else
      a4k = a4max;
   end
end

function a4k = DetermineA4k3(a4imin, a4imax, a4jmin, a4jmax, a4kmin, a4kmax, a4min,
a4max);

if min([a4imin, a4jmin, a4kmin]) < a4min
   a4min = min([a4imin, a4jmin, a4kmin]);
end
if max([a4imax, a4jmax, a4kmax]) > a4max
   a4max = max([a4imax, a4jmax, a4kmax]);
end
if ( sign(a4min) == sign(a4max)  )
   a4k = 0;
else
   if abs(a4min) <= abs(a4max)
      a4k = a4min;
   else
      a4k = a4max;
   end
end

function a4k = DetermineA4k(a4imin, a4imax);

if (sign(a4imin) == sign(a4imax))
   a4k = 0;
else
   if abs(a4imin) <= abs(a4imax)
      a4k = a4imin;
   else
```

```
      a4k = a4imax;
   end
end

function [tmin, tmax] = TimeCheck(x, z10, k, index, v, Ts, C, R, r, T);
%This function is used to find two boundary time constraints: tmin, tmax

   if ((C-v(k)) == 0) v(k) = v(k) - 0.0001;
   end

   time1 =( x(index) - z10 - r - R - v(k)*Ts(k) )/( C-v(k) );
   time2 =( x(index) - z10 + r + R - v(k)*Ts(k) )/( C-v(k) );

   if time2>time1
      if time1 < Ts(k)
        tmin = Ts(k);
      else
        tmin = time1;
      end
      if time2 > T
        tmax = T;
      else
        tmax = time2;
      end
   else
      if time2 < Ts(k)
         tmin = Ts(k);
      else
         tmin = time2;
      end
      if time1 > T
         tmax = T;
      else
         tmax = time1;
      end
   end
```

APPENDIX B: ROBOTS' CODES FOR OOPIC-R

Main Robot's code

```
'Variables and Memories Allocation

'Value stored to be transfer to the Stem as setpoint speed
'for both left and right wheels

Dim VR1    As  New   oByte
Dim VR2    As  New   oByte
Dim VR3    As  New   oByte
Dim VR4    As  New   oByte
Dim VL1    As  New   oByte
Dim VL2    As  New   oByte
Dim VL3    As  New   oByte
Dim VL4    As  New   oByte

'Temporary values transfer from the pc to the oopic-r
'these value will be store in V1 up to V8.  T9 is not used.

Dim T1    As  New   oByte
Dim T2    As  New   oByte
Dim T3    As  New   oByte
Dim T4    As  New   oByte
Dim T5    As  New   oByte
Dim T6    As  New   oByte
Dim T7    As  New   oByte
Dim T8    As  New   oByte
Dim T9    As  New   oByte

'condition varible used to communicate with the stem

Dim ReadAllow    As  New oByte

'number of 8-byte set expected to receive from the pc

Dim Count        As  New oByte

'Neccessary variables needed for communication between the oopic-r and fwcm.

Dim FWCM      As New oi2c
Dim LocNode   As New oByte  'Local node address
Dim RemNode   As New oByte  'Remote node address to write to
Dim ReadAdr   As New oByte          'Node address of remote node value came from
Dim Error     As New oBit
Dim OError    As New oBit
Dim ReadOK    As New oBit
Dim Done      As New oBit
Dim TOut      As New oNibble

'LED is used for human to see that data transfer has complete.

Dim LED       As New oDio1
Dim MoveAllow As New oByte



Sub Main()

 led.ioline = 6
 led.direction = cvOutput
```

```
led.value = 0
MoveAllow = 0


oopic.delay = 200              'Wait 2 seconds for FWCM to initialise

OOPic.Node = 4


LocNode = 2                  'Define robot node
RemNode = 7                  'Define pc node
Const WriteTOut = 8          'Define write timeout in seconds (0-15)
Const FWCMAdr = &h61  'FWCM A0 & A1 jumpers ON (Range &h60-&h63)
Call SetUpFWCM                       'Setup FWCM for communication


Count = 1
ReadAllow = 0

Do

 '1st 8 bytes
 call data 'after call data, T1 to T8 are data received from the pc.

 VR1 = T1 - 1
 VR2 = T2 - 1
 VR3 = T3 - 1
 VR4 = T4 - 1
 VL1 = T5 - 1
 VL2 = T6 - 1
 VL3 = T7 - 1
 VL4 = T8 - 1

 ReadAllow = 1
 oopic.delay = 10

 Do

 Loop Until (ReadAllow = 0)

 Count = Count + 1

Loop Until (Count = 11)


Do
call data
Loop Until ( (T1=1) and (T2=2) and (T3=4) and (T4=8) and (T5=16) and (T6=32) and (T7=64) and (T8=128) )

MoveAllow = 1


led.value = 1

End Sub

Sub SetUpFWCM()

        'Set the DS-FWCM I2C address shifted right by 1 bit
```

```
        FWCM.Node = FWCMAdr             'Setup I2C address for FWCM

        'Setup I2C addressing to FWCM

        FWCM.Width = cv8bit             'Control Info is 1-byte
        FWCM.Mode  = cv10bit            'I2C mode is 10-Bit Addressing
        FWCM.NoInc = cvFalse            'Increment on every read/write
        FWCM.Location = 0                       'Specifiy local node address location
        FWCM.Value = LocNode            'and write local node address

End Sub
'
' Subroutine to write data to remote FWCM (RemNode) address and upon
' confirmation that data has been written store remote node
' ADC values in RemADC0 - RemADC3
' On exit 'Error' = False if write completed OK
'
Sub WriteFWCM()


    FWCM.Location = 7           'Set the FWCM to write to extended registers
    'FWCM.Value = WriteVal2      '2nd byte of data to remote node
    'FWCM.Value = WriteVal3
    'FWCM.Value = WriteVal4
    'FWCM.Value = WriteVal5
    'FWCM.Value = WriteVal6
    'FWCM.Value = WriteVal7
    'FWCM.Value = WriteVal8
    'FWCM.Value = WriteVal9      '9th byte of data to remote node
        oopic.delay = 10
        FWCM.Location = 0       '1                 'Start at remote node address
        FWCM.Value = LocNode
        FWCM.Value = RemNode                    'Set remote node address to write to
        FWCM.Value = WriteTOut                  'Set write timeout value 1-15 seconds
        'FWCM.Value = WriteVal1                 'Send 1st byte of data to remote node

        call    WaitFWCM                        'Wait for confirmation or error

End Sub
'
' Subroutine to check for data from remote FWCM(s) and upon
' confirmation that data has arrived store data in 'ReadVal' and
' remote FWCM address in 'ReadAdr'.
' On exit 'ReadOK' = True if data was read and 'OError' = True
' if internal data register has overflowed i.e. one or more data
' bytes may have been lost
'
Sub ReadFWCM()

FWCM.Location = 2                       'Start at read status flag (R2)
If FWCM.Value = 255 then 'Has data been received (R2) ?
 If FWCM.Value  = 0 then            'If so has overflow error occured (R3) ?
   ReadAdr  = FWCM.Value            'If not then store FWCM address data came from
   T1  = FWCM.Value         'and store data
   FWCM.Location = 20       'Set the FWCM to read back from register 20
   T2  = FWCM.Value         'Read reg 20 and put in byte
   T3  = FWCM.Value         'Read reg 21 and put in byte
   T4  = FWCM.Value         'Read reg 22 and put in byte
   T5  = FWCM.Value         'Read reg 23 and put in byte
```

```
   T6  = FWCM.Value         'Read reg 24 and put in byte
   T7  = FWCM.Value         'Read reg 25 and put in byte
   T8  = FWCM.Value         'Read reg 26 and put in byte
   T9  = FWCM.Value         'Read reg 27 and put in byte
   OError   = cvFalse       'Indicate no overflow error
   ReadOK   = cvTrue                'Indicate data has been read and stored
 else
   OError   = cvTrue                'Indicate overflow error has occured
   ReadOK   = cvTrue                'Indicate data has been read
   ReadAdr  = FWCM.Value            'Store FWCM address data came from
   VR1  = FWCM.Value       'and store data
 end if
else
  OError = cvFalse                  'If no data read then no error
  ReadOK = cvFalse                  'and no data
end if


End Sub

' Subroutine to wait for confirmation that data has been successfully sent to
' the remote node or that a timeout error has occured

Sub WaitFWCM()

Done = cvFalse
do
 FWCM.Location = 0              'Start at write completion status flag (R0)
 If FWCM.Value = 255 then   'Has write been completed (R0) ?
  If FWCM.Value = 0 then    'Has write resulted in error (R1) ?
     Error = cvFalse                'Indicate no error has resulted
     Done = cvTrue    'and request exit from sub-routine
  else
     Error = cvTrue     'If so then indicate error
     Done = cvTrue    'and request exit from sub-routine
  end if
 else

 If FWCM.Value = 255 then           'Has write resulted in error (R1) ?
    Error = cvTrue              'If so then indicate error
    Done = cvTrue                   'and request exit from sub-routine
 end if
end if
loop until Done = cvTrue
End Sub



Sub data()
'read 9 bytes send by a pc
 do
 call       ReadFWCM
 loop until ReadOK = cvTrue
'After performing ReadFWCM, T1...T8 are stored
End Sub
```

Obstacle 1's code

```
Dim T1    As  New   oByte
Dim T2    As  New   oByte
Dim T3    As  New   oByte
```

```
Dim T4    As  New   oByte
Dim T5    As  New   oByte
Dim T6    As  New   oByte
Dim T7    As  New   oByte
Dim T8    As  New   oByte
Dim T9    As  New   oByte

Dim FWCM      As New oi2c
Dim LocNode   As New oByte  'Local node address
Dim RemNode   As New oByte  'Remote node address to write to
Dim ReadAdr   As New oByte          'Node address of remote node value came from
Dim Error     As New oBit
Dim OError    As New oBit
Dim ReadOK    As New oBit
Dim Done      As New oBit
Dim TOut      As New oNibble
Dim LED1      As New oDio1
Dim LED2      As New oDio1
Dim LED3      As New oDio1

Dim A         As New oFreq
Dim B         As New oFreq

Dim RWheel    As New oDCMotor2
Dim LWheel    As New oDCMotor2

Dim REncoder  As New oQencode
Dim LEncoder  As New oQencode

Dim LTen      As New oByte
Dim LRemainder As New oByte
Dim LCounter  As New oByte

Dim RTen      As New oByte
Dim RRemainder As New oByte
Dim RCounter  As New oByte
Dim Speed     As New oByte

Dim Button    As New oDio1



Sub Main()


oopic.delay = 200  'Wait 2 seconds for FWCM to initialise

SetLED
SetWheel

 LocNode = 1                'Define robot node
 RemNode = 7                'Define pc node
 Const WriteTOut = 8        'Define write timeout in seconds (0-15)
 Const FWCMAdr = &h61  'FWCM A0 & A1 jumpers ON (Range &h60-&h63)
 Call SetUpFWCM                     'Setup FWCM for communication

Do
call data
Loop Until ( (T1=1) and (T2=2) and (T3=4) and (T4=8) and (T5=16) and (T6=32) and (T7=64) and (T8=128) )
```

```
LED1.value = 1
LED2.value = 1
LED3.value = 1

'Button.IOLine = 6
'Button.Direction = cvInput
'Button = 0

'Do
'Loop Until (Button = 1)

call speed2_4
oopic.delay = 900

turnright '56 delay

call speed3_23
oopic.delay = 900

turnleft '19 delay
stop
oopic.delay = 29

call speed2_83
oopic.delay = 500
oopic.delay = 500
oopic.delay = 500

call stop




End Sub



Sub SetUpFWCM()
        'Set the DS-FWCM I2C address shifted right by 1 bit
        FWCM.Node = FWCMAdr             'Setup I2C address for FWCM
        'Setup I2C addressing to FWCM
        FWCM.Width = cv8bit             'Control Info is 1-byte
        FWCM.Mode  = cv10bit            'I2C mode is 10-Bit Addressing
        FWCM.NoInc = cvFalse            'Increment on every read/write
        FWCM.Location = 0                       'Specifiy local node address location
        FWCM.Value = LocNode            'and write local node address
End Sub

' Subroutine to write data to remote FWCM (RemNode) address and upon
' confirmation that data has been written store remote node
' ADC values in RemADC0 - RemADC3
' On exit 'Error' = False if write completed OK
'
Sub WriteFWCM()
    FWCM.Location = 7           'Set the FWCM to write to extended registers
    'FWCM.Value = WriteVal2      '2nd byte of data to remote node
    'FWCM.Value = WriteVal3
    'FWCM.Value = WriteVal4
```

```
    'FWCM.Value = WriteVal5
    'FWCM.Value = WriteVal6
    'FWCM.Value = WriteVal7
    'FWCM.Value = WriteVal8
    'FWCM.Value = WriteVal9       '9th byte of data to remote node
        oopic.delay = 10
        FWCM.Location = 0         '1                'Start at remote node address
        FWCM.Value = LocNode
        FWCM.Value = RemNode                        'Set remote node address to write to
        FWCM.Value = WriteTOut                      'Set write timeout value 1-15 seconds
        'FWCM.Value = WriteVal1                     'Send 1st byte of data to remote node
        call       WaitFWCM                         'Wait for confirmation or error
End Sub
'
' Subroutine to check for data from remote FWCM(s) and upon
' confirmation that data has arrived store data in 'ReadVal' and
' remote FWCM address in 'ReadAdr'.
' On exit 'ReadOK' = True if data was read and 'OError' = True
' if internal data register has overflowed i.e. one or more data
' bytes may have been lost
'
Sub ReadFWCM()
FWCM.Location = 2                           'Start at read status flag (R2)
If FWCM.Value = 255 then 'Has data been received (R2) ?
 If FWCM.Value  = 0 then              'If so has overflow error occured (R3) ?
   ReadAdr  = FWCM.Value              'If not then store FWCM address data came from
   T1  = FWCM.Value         'and store data
   FWCM.Location = 20       'Set the FWCM to read back from register 20
   T2  = FWCM.Value         'Read reg 20 and put in byte
   T3  = FWCM.Value         'Read reg 21 and put in byte
   T4  = FWCM.Value         'Read reg 22 and put in byte
   T5  = FWCM.Value         'Read reg 23 and put in byte
   T6  = FWCM.Value         'Read reg 24 and put in byte
   T7  = FWCM.Value         'Read reg 25 and put in byte
   T8  = FWCM.Value         'Read reg 26 and put in byte
   T9  = FWCM.Value         'Read reg 27 and put in byte
   OError   = cvFalse       'Indicate no overflow error
   ReadOK   = cvTrue                    'Indicate data has been read and stored
 else
   OError   = cvTrue                    'Indicate overflow error has occured
   ReadOK   = cvTrue                    'Indicate data has been read
   ReadAdr  = FWCM.Value                'Store FWCM address data came from
   T9  = FWCM.Value         'and store data
 end if
else
  OError = cvFalse                      'If no data read then no error
  ReadOK = cvFalse                      'and no data
end if
End Sub

' Subroutine to wait for confirmation that data has been successfully sent to
' the remote node or that a timeout error has occured

Sub WaitFWCM()
Done = cvFalse
do
 FWCM.Location = 0              'Start at write completion status flag (R0)
 If FWCM.Value = 255 then   'Has write been completed (R0) ?
  If FWCM.Value = 0 then    'Has write resulted in error (R1) ?
     Error = cvFalse              'Indicate no error has resulted
```

```
    Done = cvTrue    'and request exit from sub-routine
  else
    Error = cvTrue    'If so then indicate error
    Done = cvTrue    'and request exit from sub-routine
  end if
 else
 If FWCM.Value = 255 then            'Has write resulted in error (R1) ?
   Error = cvTrue              'If so then indicate error
   Done = cvTrue                       'and request exit from sub-routine
 end if
end if
loop until Done = cvTrue
End Sub



Sub data()
'read 9 bytes send by a pc
 do
 call       ReadFWCM
 loop until ReadOK = cvTrue
'After performing ReadFWCM, T1...T8 are stored
End Sub

sub speed2_4()
LWheel.Direction = 1
RWheel.Direction = 1
LWheel = 71
RWheel = 67
RWheel.Operate = 1
LWheel.Operate = 1
End Sub

sub speed3_23()
LWheel.Direction = 1
RWheel.Direction = 1
LWheel = 73
RWheel = 71
RWheel.Operate = 1
LWheel.Operate = 1
End Sub

sub speed2_83()
LWheel.Direction = 1
RWheel.Direction = 1
LWheel = 70
RWheel = 68
RWheel.Operate = 1
LWheel.Operate = 1
End Sub



sub turnleft()

LWheel.Direction = 1
RWheel.Direction = 1
LWheel = 0
RWheel = 100
RWheel.Operate = 1
```

```
LWheel.Operate = 1
    oopic.delay = 19  '272 delay turn left 360
'turn left 25.68 degrees

end sub


sub turnright()
RWheel.Direction = 1
LWheel.Direction = 1
LWheel = 100
RWheel = 0
RWheel.Operate = 1
LWheel.Operate = 1
    oopic.delay = 56 '286 delay turn right 360
'turn right 70 degrees
end sub


Sub SetWheel()
  RWheel.IOLineP = 18
  RWheel.IOLine1 = 26
  RWheel.IOLine2 = 27
  LWheel.IOLineP = 17
  LWheel.IOLine1 = 24
  LWheel.IOLine2 = 25
  RWheel.Unsigned  = 1
  RWheel.PreScale  = 1
  RWheel.Period    = 100
  RWheel.Direction = 1
  RWheel.Brake = cvOff
  LWheel.Unsigned  = 1
  LWheel.PreScale  = 1
  LWheel.Period    = 100
  LWheel.Direction = 1
  LWheel.Brake = cvOff
End Sub


Sub SetLED()
 LED1.IOLine = 5
 LED1.direction = cvOutput
 LED1.value = 0
 LED2.IOLine = 6
 LED2.direction = cvOutput
 LED2.value = 0
 LED3.IOLine = 7
 LED3.direction = cvOutput
 LED3.value = 0
End Sub

Sub stop()
RWheel.Operate = 0
LWheel.Operate = 0
End Sub
```

Obstacle 2's code

```
Dim T1    As  New   oByte
```

```
Dim T2    As  New   oByte
Dim T3    As  New   oByte
Dim T4    As  New   oByte
Dim T5    As  New   oByte
Dim T6    As  New   oByte
Dim T7    As  New   oByte
Dim T8    As  New   oByte
Dim T9    As  New   oByte

Dim FWCM      As New oi2c
Dim LocNode   As New oByte  'Local node address
Dim RemNode   As New oByte  'Remote node address to write to
Dim ReadAdr   As New oByte          'Node address of remote node value came from
Dim Error     As New oBit
Dim OError    As New oBit
Dim ReadOK    As New oBit
Dim Done      As New oBit
Dim TOut      As New oNibble
Dim LED1      As New oDio1
Dim LED2      As New oDio1
Dim LED3      As New oDio1

Dim A         As New oFreq
Dim B         As New oFreq

Dim RWheel    As New oDCMotor2
Dim LWheel    As New oDCMotor2

Dim REncoder  As New oQencode
Dim LEncoder  As New oQencode

Dim LTen      As New oByte
Dim LRemainder As New oByte
Dim LCounter  As New oByte

Dim RTen      As New oByte
Dim RRemainder As New oByte
Dim RCounter  As New oByte
Dim Speed     As New oByte
Dim Button    As New oDio1


Sub Main()



 'Button.IOLine = 6
 SetLED
 SetWheel

 oopic.delay = 200             'Wait 2 seconds for FWCM to initialise
 LocNode = 2                   'Define robot node
 RemNode = 7                   'Define pc node
 Const WriteTOut = 8           'Define write timeout in seconds (0-15)
 Const FWCMAdr = &h61  'FWCM A0 & A1 jumpers ON (Range &h60-&h63)
 Call SetUpFWCM                        'Setup FWCM for communication



Do
```

```
call data
Loop Until ( (T1=1) and (T2=2) and (T3=4) and (T4=8) and (T5=16) and (T6=32) and (T7=64) and (T8=128) )

LED1.value = 1
LED2.value = 1
LED3.value = 1

'Button.Direction = cvInput
'Do
'Loop Until (Button = 1)
'oopic.delay = 200

speed4_1
oopic.delay = 600

speed3_0
oopic.delay = 300

turnleft  'delay = 7

stop
oopic.delay = 49

speed3_1
oopic.delay = 300

speed3_65
oopic.delay = 600


stop
oopic.delay = 48

speed3_65

oopic.delay = 400
oopic.delay = 500
oopic.delay = 500
oopic.delay = 400
'oopic.delay = 200
stop
 End Sub


Sub SetUpFWCM()
        'Set the DS-FWCM I2C address shifted right by 1 bit
        FWCM.Node = FWCMAdr             'Setup I2C address for FWCM
        'Setup I2C addressing to FWCM
        FWCM.Width = cv8bit             'Control Info is 1-byte
        FWCM.Mode  = cv10bit            'I2C mode is 10-Bit Addressing
        FWCM.NoInc = cvFalse            'Increment on every read/write
        FWCM.Location = 0                       'Specifiy local node address location
        FWCM.Value = LocNode            'and write local node address
End Sub
'
' Subroutine to write data to remote FWCM (RemNode) address and upon
' confirmation that data has been written store remote node
' ADC values in RemADC0 - RemADC3
' On exit 'Error' = False if write completed OK
'
```

```
Sub WriteFWCM()
    FWCM.Location = 7           'Set the FWCM to write to extended registers
    'FWCM.Value = WriteVal2     '2nd byte of data to remote node
    'FWCM.Value = WriteVal3
    'FWCM.Value = WriteVal4
    'FWCM.Value = WriteVal5
    'FWCM.Value = WriteVal6
    'FWCM.Value = WriteVal7
    'FWCM.Value = WriteVal8
    'FWCM.Value = WriteVal9     '9th byte of data to remote node
        oopic.delay = 10
        FWCM.Location = 0       '1                 'Start at remote node address
        FWCM.Value = LocNode
        FWCM.Value = RemNode                       'Set remote node address to write to
        FWCM.Value = WriteTOut                     'Set write timeout value 1-15 seconds
        'FWCM.Value = WriteVal1                    'Send 1st byte of data to remote node
        call    WaitFWCM                           'Wait for confirmation or error
End Sub
'
' Subroutine to check for data from remote FWCM(s) and upon
' confirmation that data has arrived store data in 'ReadVal' and
' remote FWCM address in 'ReadAdr'.
' On exit 'ReadOK' = True if data was read and 'OError' = True
' if internal data register has overflowed i.e. one or more data
' bytes may have been lost
'
Sub ReadFWCM()


FWCM.Location = 2                          'Start at read status flag (R2)
If FWCM.Value = 255 then 'Has data been received (R2) ?
 If FWCM.Value  = 0 then            'If so has overflow error occured (R3) ?
  ReadAdr  = FWCM.Value             'If not then store FWCM address data came from
  T1  = FWCM.Value        'and store data
  FWCM.Location = 20      'Set the FWCM to read back from register 20
  T2  = FWCM.Value        'Read reg 20 and put in byte
  T3  = FWCM.Value        'Read reg 21 and put in byte
  T4  = FWCM.Value        'Read reg 22 and put in byte
  T5  = FWCM.Value        'Read reg 23 and put in byte
  T6  = FWCM.Value        'Read reg 24 and put in byte
  T7  = FWCM.Value        'Read reg 25 and put in byte
  T8  = FWCM.Value        'Read reg 26 and put in byte
  T9  = FWCM.Value        'Read reg 27 and put in byte
  OError   = cvFalse      'Indicate no overflow error
  ReadOK   = cvTrue                 'Indicate data has been read and stored
 else
  OError   = cvTrue                 'Indicate overflow error has occured
  ReadOK   = cvTrue                 'Indicate data has been read
  ReadAdr  = FWCM.Value             'Store FWCM address data came from
  T9  = FWCM.Value        'and store data
 end if
else
  OError = cvFalse                  'If no data read then no error
  ReadOK = cvFalse                  'and no data
end if
End Sub

' Subroutine to wait for confirmation that data has been successfully sent to
' the remote node or that a timeout error has occured
```

```
Sub WaitFWCM()
Done = cvFalse
do
 FWCM.Location = 0              'Start at write completion status flag (R0)
 If FWCM.Value = 255 then   'Has write been completed (R0) ?
  If FWCM.Value = 0 then    'Has write resulted in error (R1) ?
     Error = cvFalse                'Indicate no error has resulted
     Done = cvTrue    'and request exit from sub-routine
  else
     Error = cvTrue    'If so then indicate error
     Done = cvTrue    'and request exit from sub-routine
  end if
 else

 If FWCM.Value = 255 then             'Has write resulted in error (R1) ?
    Error = cvTrue              'If so then indicate error
    Done = cvTrue                       'and request exit from sub-routine
 end if
end if
loop until Done = cvTrue
End Sub



Sub data()
'read 9 bytes send by a pc
 do
 call        ReadFWCM
 loop until ReadOK = cvTrue
'After performing ReadFWCM, T1...T8 are stored
End Sub


Sub Turnleft()
LWheel.Direction = 0
RWheel.Direction = 1
LWheel = 0
RWheel = 100
RWheel.Operate = 1
LWheel.Operate = 1
     oopic.delay = 7
                    '258 is 360 degrees turn
End Sub


Sub Stop()
     RWheel.Operate = 0  'stop the wheel and check distance traveled
     LWheel.Operate = 0
End Sub


Sub SetWheel()
  RWheel.IOLineP = 18
  RWheel.IOLine1 = 26
  RWheel.IOLine2 = 27
  LWheel.IOLineP = 17
  LWheel.IOLine1 = 24
  LWheel.IOLine2 = 25
  RWheel.Unsigned  = 1
```

```
  RWheel.PreScale  = 1
  RWheel.Period    = 100
  RWheel.Direction = 1
  RWheel.Brake = cvOff
  LWheel.Unsigned  = 1
  LWheel.PreScale  = 1
  LWheel.Period    = 100
  LWheel.Direction = 1
  LWheel.Brake = cvOff
End Sub


Sub SetLED()
 LED1.IOLine = 5
 LED1.direction = cvOutput
 LED1.value = 0

 LED2.IOLine = 6
 LED2.direction = cvOutput
 LED2.value = 0

 LED3.IOLine = 7
 LED3.direction = cvOutput
 LED3.value = 0
End Sub

Sub speed4_1()
LWheel.Direction = 0
RWheel.Direction = 0
LWheel = 73
RWheel = 68
RWheel.Operate = 1
LWheel.Operate = 1
End Sub

Sub speed3_1()
LWheel.Direction = 1
RWheel.Direction = 1
LWheel = 67
RWheel = 61
RWheel.Operate = 1
LWheel.Operate = 1
End Sub


Sub speed3_65()
LWheel.Direction = 1
RWheel.Direction = 1
LWheel = 70
RWheel = 65
RWheel.Operate = 1
LWheel.Operate = 1
End Sub

Sub speed3_0()
LWheel.Direction = 0
RWheel.Direction = 0
LWheel = 67
RWheel = 62
RWheel.Operate = 1
```

```
LWheel.Operate = 1
End Sub
```

Obstacle 3's Code

```
Dim T1    As  New   oByte
Dim T2    As  New   oByte
Dim T3    As  New   oByte
Dim T4    As  New   oByte
Dim T5    As  New   oByte
Dim T6    As  New   oByte
Dim T7    As  New   oByte
Dim T8    As  New   oByte
Dim T9    As  New   oByte

Dim FWCM      As New oi2c
Dim LocNode   As New oByte  'Local node address
Dim RemNode   As New oByte  'Remote node address to write to
Dim ReadAdr   As New oByte          'Node address of remote node value came from
Dim Error     As New oBit
Dim OError    As New oBit
Dim ReadOK    As New oBit
Dim Done      As New oBit
Dim TOut      As New oByte
Dim LED1      As New oDio1
Dim LED2      As New oDio1
Dim LED3      As New oDio1

Dim A         As New oFreq
Dim B         As New oFreq

Dim RWheel    As New oDCMotor2
Dim LWheel    As New oDCMotor2

Dim REncoder   As New oQencode
Dim LEncoder   As New oQencode

Dim LTen       As New oByte
Dim LRemainder As New oByte
Dim LCounter   As New oByte

Dim RTen       As New oByte
Dim RRemainder As New oByte
Dim RCounter   As New oByte
Dim Speed      As New oByte



Dim Button     As New oDio1



Sub Main()

 SetLED
 SetWheel
' Button.IOLine = 6
```

```
' Button.Direction = cvInput


 oopic.delay = 200              'Wait 2 seconds for FWCM to initialise

 LocNode = 3                   'Define robot node
 RemNode = 7                   'Define pc node
 Const WriteTOut = 8           'Define write timeout in seconds (0-15)
 Const FWCMAdr = &h61  'FWCM A0 & A1 jumpers ON (Range &h60-&h63)
 Call SetUpFWCM                        'Setup FWCM for communication



Do
call data
Loop Until ( (T1=1) and (T2=2) and (T3=4) and (T4=8) and (T5=16) and (T6=32) and (T7=64) and (T8=128) )


LED1.value = 1
LED2.value = 1
LED3.value = 1

'Button = 0

'Do
'Loop Until (Button = 1)

'oopic.delay = 200

speed1_19
oopic.delay = 900


stop
oopic.delay = 56

speed1_19
oopic.delay = 900

turnright '90 degrees


speed1_19
oopic.delay = 500
oopic.delay = 500
oopic.delay = 500
oopic.delay = 300

stop


End Sub



Sub SetUpFWCM()

        'Set the DS-FWCM I2C address shifted right by 1 bit

        FWCM.Node = FWCMAdr             'Setup I2C address for FWCM
```

```
        'Setup I2C addressing to FWCM

        FWCM.Width = cv8bit             'Control Info is 1-byte
        FWCM.Mode  = cv10bit            'I2C mode is 10-Bit Addressing
        FWCM.NoInc = cvFalse            'Increment on every read/write
        FWCM.Location = 0                       'Specifiy local node address location
        FWCM.Value = LocNode            'and write local node address

End Sub
'
' Subroutine to write data to remote FWCM (RemNode) address and upon
' confirmation that data has been written store remote node
' ADC values in RemADC0 - RemADC3
' On exit 'Error' = False if write completed OK
'
Sub WriteFWCM()


    FWCM.Location = 7           'Set the FWCM to write to extended registers
    'FWCM.Value = WriteVal2      '2nd byte of data to remote node
    'FWCM.Value = WriteVal3
    'FWCM.Value = WriteVal4
    'FWCM.Value = WriteVal5
    'FWCM.Value = WriteVal6
    'FWCM.Value = WriteVal7
    'FWCM.Value = WriteVal8
    'FWCM.Value = WriteVal9      '9th byte of data to remote node
        oopic.delay = 10
        FWCM.Location = 0       '1                 'Start at remote node address
        FWCM.Value = LocNode
        FWCM.Value = RemNode                    'Set remote node address to write to
        FWCM.Value = WriteTOut                  'Set write timeout value 1-15 seconds
        'FWCM.Value = WriteVal1                 'Send 1st byte of data to remote node

        call    WaitFWCM                        'Wait for confirmation or error

End Sub
'
' Subroutine to check for data from remote FWCM(s) and upon
' confirmation that data has arrived store data in 'ReadVal' and
' remote FWCM address in 'ReadAdr'.
' On exit 'ReadOK' = True if data was read and 'OError' = True
' if internal data register has overflowed i.e. one or more data
' bytes may have been lost
'
Sub ReadFWCM()

FWCM.Location = 2                               'Start at read status flag (R2)
If FWCM.Value = 255 then 'Has data been received (R2) ?
 If FWCM.Value  = 0 then            'If so has overflow error occured (R3) ?
   ReadAdr  = FWCM.Value            'If not then store FWCM address data came from
   T1  = FWCM.Value         'and store data
   FWCM.Location = 20       'Set the FWCM to read back from register 20
   T2  = FWCM.Value         'Read reg 20 and put in byte
   T3  = FWCM.Value         'Read reg 21 and put in byte
   T4  = FWCM.Value         'Read reg 22 and put in byte
   T5  = FWCM.Value         'Read reg 23 and put in byte
   T6  = FWCM.Value         'Read reg 24 and put in byte
   T7  = FWCM.Value         'Read reg 25 and put in byte
```

```
   T8  = FWCM.Value         'Read reg 26 and put in byte
   T9  = FWCM.Value         'Read reg 27 and put in byte
   OError   = cvFalse       'Indicate no overflow error
   ReadOK   = cvTrue                'Indicate data has been read and stored
 else
   OError   = cvTrue                'Indicate overflow error has occured
   ReadOK   = cvTrue                'Indicate data has been read
   ReadAdr  = FWCM.Value            'Store FWCM address data came from
   T9  = FWCM.Value         'and store data
 end if
else
  OError = cvFalse                  'If no data read then no error
  ReadOK = cvFalse                  'and no data
end if


End Sub

' Subroutine to wait for confirmation that data has been successfully sent to
' the remote node or that a timeout error has occured

Sub WaitFWCM()

Done = cvFalse
do
 FWCM.Location = 0              'Start at write completion status flag (R0)
 If FWCM.Value = 255 then   'Has write been completed (R0) ?
  If FWCM.Value = 0 then    'Has write resulted in error (R1) ?
     Error = cvFalse                'Indicate no error has resulted
     Done = cvTrue    'and request exit from sub-routine
  else
     Error = cvTrue     'If so then indicate error
     Done = cvTrue    'and request exit from sub-routine
  end if
 else

 If FWCM.Value = 255 then           'Has write resulted in error (R1) ?
    Error = cvTrue              'If so then indicate error
    Done = cvTrue                       'and request exit from sub-routine
 end if
end if
loop until Done = cvTrue
End Sub



Sub data()
'read 9 bytes send by a pc
 do
 call        ReadFWCM
 loop until ReadOK = cvTrue
'After performing ReadFWCM, T1...T8 are stored
End Sub

Sub SetLED()
 LED1.IOLine = 5
 LED1.direction = cvOutput
 LED1.value = 0
 LED2.IOLine = 6
 LED2.direction = cvOutput
```

```
 LED2.value = 0
 LED3.IOLine = 7
 LED3.direction = cvOutput
 LED3.value = 0
End Sub


Sub Stop()
    RWheel.Operate = 0  'stop the wheel and check distance traveled
    LWheel.Operate = 0
End sub


Sub TurnLeft()
LWheel.Direction = 0
RWheel.Direction = 1
LWheel = 80  '60
RWheel = 80  '68
RWheel.Operate = 1
LWheel.Operate = 1
oopic.delay = 146 '180 make a robot turn 360 left
End Sub


Sub Turnright()
LWheel.Direction = 1
RWheel.Direction = 1
LWheel = 100  '60
RWheel = 0  '68
RWheel.Operate = 1
LWheel.Operate = 1
oopic.delay = 48 '193 make a robot turns 360 right
End Sub


Sub speed1_19()
LWheel.Direction = 1
RWheel.Direction = 1
RWheel = 60  '68
LWheel = 60  '60
RWheel.Operate = 1
LWheel.Operate = 1
oopic.delay = 1
LWheel = 29
RWheel = 57
End Sub


Sub SetWheel()
  RWheel.IOLineP = 18
  RWheel.IOLine1 = 26
  RWheel.IOLine2 = 27
  LWheel.IOLineP = 17
  LWheel.IOLine1 = 24
  LWheel.IOLine2 = 25

  RWheel.Unsigned  = 1
  RWheel.PreScale  = 1
  RWheel.Period    = 100
  RWheel.Direction = 1
```

```
  RWheel.Brake = cvOff

  LWheel.Unsigned  = 1
  LWheel.PreScale  = 1
  LWheel.Period    = 100
  LWheel.Direction = 1
  LWheel.Brake = cvOff
End Sub
```

APPENDIX C: ROBOT'S CODE FOR BRAINSTEM

```
/* Test.tea */
/* tests 1-byte transfers between BrainStem and OOPic */
#include <aCore.tea>
#include <aPrint.tea>
#include <aMotion.tea>
#defineADDR_OOPIC          8


void aOOPic_WriteChar(char oopic, char addr, char mask, char data)
{
  asm
  {
    pushsb 6      /* OOPic address */
    pushlb 3      /* command data size */
    pushsb 7      /* data address */
    pushsb 7      /* bit mask */
    pushsb 7      /* data */
    pushlb 5      /* total command size */
    popcmd        /* send command */
  }
}

char aOOPic_ReadChar(char oopic, char addr)
{
  char c=0;
  asm
  {
    pushsb 5      /* OOPic address */
    pushlb 1      /* command data size */
    pushsb 6      /* data address */
    pushlb 3      /* total command size */
    popcmd        /* send command */

    pushsb 5      /* OOPic address */
    pushlb 1      /* set low bit for read */
    orb
    pushlb 1      /* number of bytes to read */
    popsm aPortIICRead          /* do the read */

    popbs 1       /* set return value */
  }
  return c;
}

void main()
```

```
{
char l1, l2, l3, l4, l5, l6, l7, l8, l9, l10, l11, l12, l13, l14, l15, l16, l17, l18, l19, l20;
char l21, l22, l23, l24, l25, l26, l27, l28, l29, l30, l31, l32, l33, l34, l35, l36, l37, l38, l39, l40;
char r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20;
char r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31, r32, r33, r34, r35, r36, r37, r38, r39, r40;
char stop, ReadAllow, MoveAllow;

  stop = 0;
  ReadAllow = 0;
  MoveAllow = 0;


  aCore_Sleep(10000);



  ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);


   while (ReadAllow != 1)
{
    aCore_Sleep(100);
    ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}




    r1=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
    r2=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
    r3=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
    r4=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
    l1=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
    l2=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
    l3=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
    l4=aOOPic_ReadChar(ADDR_OOPIC,(char)241);

aCore_Sleep(100);
aOOPic_WriteChar(ADDR_OOPIC,(char)221,0,0);
aCore_Sleep(100);

  ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);
```

```
  while (ReadAllow != 1)
{
   aCore_Sleep(100);
   ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}


   r5=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
   r6=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
   r7=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
   r8=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
   l5=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
   l6=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
   l7=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
   l8=aOOPic_ReadChar(ADDR_OOPIC,(char)241);

aCore_Sleep(100);
aOOPic_WriteChar(ADDR_OOPIC,(char)221,0,0);
aCore_Sleep(100);

 ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);


  while (ReadAllow != 1)
{
   aCore_Sleep(100);
   ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}


   r9=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
   r10=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
   r11=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
   r12=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
   l9=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
   l10=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
   l11=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
   l12=aOOPic_ReadChar(ADDR_OOPIC,(char)241);


aCore_Sleep(100);
aOOPic_WriteChar(ADDR_OOPIC,(char)221,0,0);
aCore_Sleep(100);
```

```
  ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);


   while (ReadAllow != 1)
{
    aCore_Sleep(100);
    ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}


    r13=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
    r14=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
    r15=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
    r16=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
    l13=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
    l14=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
    l15=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
    l16=aOOPic_ReadChar(ADDR_OOPIC,(char)241);



aCore_Sleep(100);
aOOPic_WriteChar(ADDR_OOPIC,(char)221,0,0);
aCore_Sleep(100);

  ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);


   while (ReadAllow != 1)
{
    aCore_Sleep(100);
    ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}


    r17=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
    r18=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
    r19=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
    r20=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
    l17=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
    l18=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
    l19=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
```

```
   l20=aOOPic_ReadChar(ADDR_OOPIC,(char)241);


aCore_Sleep(100);
aOOPic_WriteChar(ADDR_OOPIC,(char)221,0,0);
aCore_Sleep(100);

  ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);


   while (ReadAllow != 1)
{
   aCore_Sleep(100);
   ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}


   r21=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
   r22=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
   r23=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
   r24=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
   l21=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
   l22=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
   l23=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
   l24=aOOPic_ReadChar(ADDR_OOPIC,(char)241);




aCore_Sleep(100);
aOOPic_WriteChar(ADDR_OOPIC,(char)221,0,0);
aCore_Sleep(100);

  ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);


   while (ReadAllow != 1)
{
   aCore_Sleep(100);
   ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}
```

```
   r25=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
   r26=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
   r27=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
   r28=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
   l25=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
   l26=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
   l27=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
   l28=aOOPic_ReadChar(ADDR_OOPIC,(char)241);



aCore_Sleep(100);
aOOPic_WriteChar(ADDR_OOPIC,(char)221,0,0);
aCore_Sleep(100);

  ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);


   while (ReadAllow != 1)
{
   aCore_Sleep(100);
   ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}


   r29=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
   r30=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
   r31=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
   r32=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
   l29=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
   l30=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
   l31=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
   l32=aOOPic_ReadChar(ADDR_OOPIC,(char)241);


aCore_Sleep(100);
aOOPic_WriteChar(ADDR_OOPIC,(char)221,0,0);
aCore_Sleep(100);

  ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);


   while (ReadAllow != 1)
```

```
{
   aCore_Sleep(100);
   ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}


   r33=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
   r34=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
   r35=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
   r36=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
   l33=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
   l34=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
   l35=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
   l36=aOOPic_ReadChar(ADDR_OOPIC,(char)241);




aCore_Sleep(100);
aOOPic_WriteChar(ADDR_OOPIC,(char)221,0,0);
aCore_Sleep(100);

  ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);


   while (ReadAllow != 1)
{
   aCore_Sleep(100);
   ReadAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}


   r37=aOOPic_ReadChar(ADDR_OOPIC,(char)255);
   r38=aOOPic_ReadChar(ADDR_OOPIC,(char)253);
   r39=aOOPic_ReadChar(ADDR_OOPIC,(char)251);
   r40=aOOPic_ReadChar(ADDR_OOPIC,(char)249);
   l37=aOOPic_ReadChar(ADDR_OOPIC,(char)247);
   l38=aOOPic_ReadChar(ADDR_OOPIC,(char)245);
   l39=aOOPic_ReadChar(ADDR_OOPIC,(char)243);
   l40=aOOPic_ReadChar(ADDR_OOPIC,(char)241);
```

```
//aCore_Sleep(10000);


  while (MoveAllow != 1)
{
   aCore_Sleep(100);
   MoveAllow = aOOPic_ReadChar(ADDR_OOPIC,(char)221);

}



//aMotion_SetVelocity(Channel 0, Channel 1);
//Channel 1 is Right Wheel,  Channel 0 is Left Wheel

//1-5

aMotion_SetVelocity(l1,r1);
aCore_Sleep(10000);

aMotion_SetVelocity(l2,r2);
aCore_Sleep(10000);

aMotion_SetVelocity(l3,r3);
aCore_Sleep(10000);

aMotion_SetVelocity(l4,r4);
aCore_Sleep(10000);

aMotion_SetVelocity(l5,r5);
aCore_Sleep(10000);

//6-10

aMotion_SetVelocity(l6,r6);
aCore_Sleep(10000);

aMotion_SetVelocity(l7,r7);
aCore_Sleep(10000);

aMotion_SetVelocity(l8,r8);
aCore_Sleep(10000);
```

```
aMotion_SetVelocity(l9,r9);
aCore_Sleep(10000);

//pause for .56 of a second at the 9th second of running
aMotion_SetVelocity(0,0);
aCore_Sleep(5600);


aMotion_SetVelocity(l10,r10);
aCore_Sleep(10000);

//11-15

aMotion_SetVelocity(l11,r11);
aCore_Sleep(10000);

aMotion_SetVelocity(l12,r12);
aCore_Sleep(10000);

aMotion_SetVelocity(l13,r13);
aCore_Sleep(10000);

aMotion_SetVelocity(l14,r14);
aCore_Sleep(10000);

aMotion_SetVelocity(l15,r15);
aCore_Sleep(10000);

//16-20

aMotion_SetVelocity(l16,r16);
aCore_Sleep(10000);

aMotion_SetVelocity(l17,r17);
aCore_Sleep(10000);

aMotion_SetVelocity(l18,r18);
aCore_Sleep(10000);

//pause for .48 of 1 second
aMotion_SetVelocity(0,0);
aCore_Sleep(4800);

aMotion_SetVelocity(l19,r19);
```

```
aCore_Sleep(10000);

aMotion_SetVelocity(l20,r20);
aCore_Sleep(10000);

//21-25

aMotion_SetVelocity(l21,r21);
aCore_Sleep(10000);

aMotion_SetVelocity(l22,r22);
aCore_Sleep(10000);

aMotion_SetVelocity(l23,r23);
aCore_Sleep(10000);

aMotion_SetVelocity(l24,r24);
aCore_Sleep(10000);

aMotion_SetVelocity(l25,r25);
aCore_Sleep(10000);

//26-30

aMotion_SetVelocity(l26,r26);
aCore_Sleep(10000);

aMotion_SetVelocity(l27,r27);
aCore_Sleep(10000);

aMotion_SetVelocity(l28,r28);
aCore_Sleep(10000);

aMotion_SetVelocity(l29,r29);
aCore_Sleep(10000);

aMotion_SetVelocity(l30,r30);
aCore_Sleep(10000);

//31-35

aMotion_SetVelocity(l31,r31);
aCore_Sleep(10000);

aMotion_SetVelocity(l32,r32);
```

```
aCore_Sleep(10000);

aMotion_SetVelocity(l33,r33);
aCore_Sleep(10000);

aMotion_SetVelocity(l34,r34);
aCore_Sleep(10000);

aMotion_SetVelocity(l35,r35);
aCore_Sleep(10000);

//36-40

aMotion_SetVelocity(l36,r36);
aCore_Sleep(10000);


aMotion_SetVelocity(l37,r37);
aCore_Sleep(10000);

aMotion_SetVelocity(l38,r38);
aCore_Sleep(10000);

aMotion_SetVelocity(l39,r39);
aCore_Sleep(10000);

aMotion_SetVelocity(l40,r40);
aCore_Sleep(10000);



aMotion_SetVelocity(stop, stop);

}
```

LIST OF REFERENCES

[1] Z. Qu, J. Wang, and C. E. Plaisted, "A New Analytical Solution to Mobile Robot Trajectory Generation in the Presence of Moving Obstacles," IEEE Transactions on Robotics and Automation, Vol. 20, pp. 978-993, December 2004.

[2] S. Monaco and D. Normand-Cyrot, "An Introduction to Motion Planning Under Multirate Digital Control," in Proceedings of the 31st Conference on Decision and Control, Tucson, Arizona, December 1992, pp. 1780-1785.

[3] R. M. Murray and S. S. Sastry, "Nonholonomic Motion Planning: Steering Using Sinusoids," IEEE Transaction on Automatic Control, Vol. 38, pp. 700-716, 1993.

[4] G. Oriolo, A. D. Luca, M. Vendittelli, "WMR Control Via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation," IEEE Transactions on Control Systems Technology, Vol. 10, No. 6, November 2002.

[5] B. Kim and P. Tsiotras, "Controllers for Unicycle-Type Wheeled Robots: Theoretical Results and Experimental Validation," IEEE Transactions on Robots and Automation, Vol. 18, No. 3, June 2002.

[6] J. Hollingworth, D. Butterfield, and B. Swart, J. Allsop, C++Builder 5, SAMS, 2000, Indianapolis, Indiana.

[7] D. Clark, Programming and Customizing the OOPic Microcontroller, McGrawHill, 2003, New York, New York.

[8] J. S. Bay, Fundamentals of Linear State Space Systems, McGrawHill, 1999, Singapore.

[9] J. Borenstein and L. Feng, “Measurement and Correction of Systematic Odometry Errors in Mobile Robots,” IEEE Journal of Robotics and Automation, May 1995.

[10] D. Tilbury, R. M. Murray, S. S. Sastry, “Trajectory generation for the N-trailer problem using Goursat normal form,” Automatic Control, Vol. 40, pp. 802-819, May 1995.