Skuba 2011 Extended Team Description

Krit Chaiso[1] and Kanjanapan Sukvichai[2]

[1] Department of Computer Engineering
[2] Department of Electrical Engineering
Faculty of Engineering, Kasetsart University
50 Phaholyothin Rd., Ladyao, Jatujak, Bangkok, 10900, Thailand
nuopolok@hotmail.com, fengkpsc@ku.ac.th

Abstract. This paper presents a detailed description of Skuba, a Small-Size League RoboCup robot team in addition to the team description paper. The robot system is designed under the RoboCup 2011 rules in order to participate in the RoboCup competition in Turkey. The low level and high level control of the SKUBA system are explained in each section.

1 Introduction

Skuba is a small-size league soccer robot team from Kasetsart University, which has entered the RoboCup competition since 2006. We got the championship from last two years from the RoboCup 2009 in Graz, Austria and 2010 in Singapore. Another championship is in April 2011 from RoboCup Iran Open 2011 in Tehran, Iran.

The robot system consists of two main components: the robot hardware and the software. The software makes strategic decisions for the robot team by using information about the object positions from the vision system. The global vision system run by the shared vision software, SSL-Vision, uses two cameras mounted over field. The software executes plans by calculating the robot actions and then sends the commands to each robot.

Our team has ten identical robots, six of them were built in 2008 and another four were built in 2009 with some minor changes in material and mechanical design. We are not planning to make any major changes to the design. The robot hardware is the same as used in last year.

This year, the main focus of this extended team description paper is to expand our control system. The control parameters are measured by automatic calibration process which simply uses the same procedure as the manual calibration does, but it's done automatically by the software.

1.1 Team Members

Kanjanapan Sukvichai : Control Theory and Supervisor
Krit Chaiso : AI Software and Team Leader
Chanon Onman : AI Software
Nuttapol Runsewa : AI Software
Khakhana Thimachai : AI Software
Phawat Lertariyasakchai : AI Software
Tanakorn Panyapiang : AI Software
Teeratath Ariyachartphadungkit : Electronics, Mechanics and Low level Firmware
Supavit Siriwan : Electronics and Mechanics
Peerapat Kittiboriluk : Mechanics

2 Robot Hardware

This section describes the robot electronics system that is used in driving system including the designs and components. Details about operations and algorithms are in the firmware section.

The robot consists of two electronics boards: the main board and the kicker board. The main board handles all of the robot tasks except kicking. The kicker board controls the entire kicker system.

2.1 Main Electronics Board

The board consists of a Xilinx Spartan-3 XC3S400 FPGA, motor driver, user interface, some add-on modules and debugging port. The microprocessor core and interfacing logic for external peripherals are implemented using FPGA in order to handle the low-level control of the brushless motor such as velocity and position control. The main electronics board receives commands from the main software on a computer. The board integrates the processing components together with the power components to keep the board compact and minimize wiring. With limited space, almost components are in small SMD packages. However, these components still large enough for hand soldering with conventional tools. Figure 1 show the main electronics board of the robot.

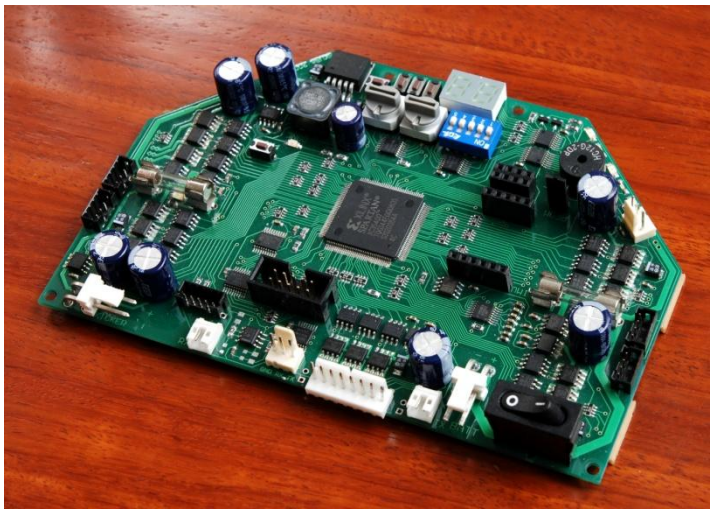


Fig. 1. The main electronics board.

2.2 Motors

There are two types of motor in the robot, the driving motor and the dribbling motor, both are brushless motor. Each driving motor is a 30 watts Maxon EC45 flat motor with a custom back-extended shaft for attaching encoder wheel. The motor itself can produce a feedback signal from hall sensors for measuring wheel velocity. However, this multi-pole motor sends only roughly 48 pulses per revolution; therefore, this motor is equipped with an US Digital E4P encoder which have higher resolution of 1440 pulses per revolution. The dribbling motor is a high speed 15 watts Maxon EC16 motor. Despite a very low resolution of 6 pulses per revolution signal from hall sensors, the implementation of the PI controller is possible when running this motor at high speeds. The maximum speed of the dribbling bar is about 13000 rpm.

The motor driver is a three phase inverter circuit using complementary N and P channel power MOSFET in each phase. This configuration doesn't require bootstrap driver as in N-channel-only configuration. These MOSFETs are driven by MOSFET driver ICs to minimize switching loss. The motor commutation and PWM generation are described in the firmware section. Figure 2 shows the three-phase brushless motor driver circuit.

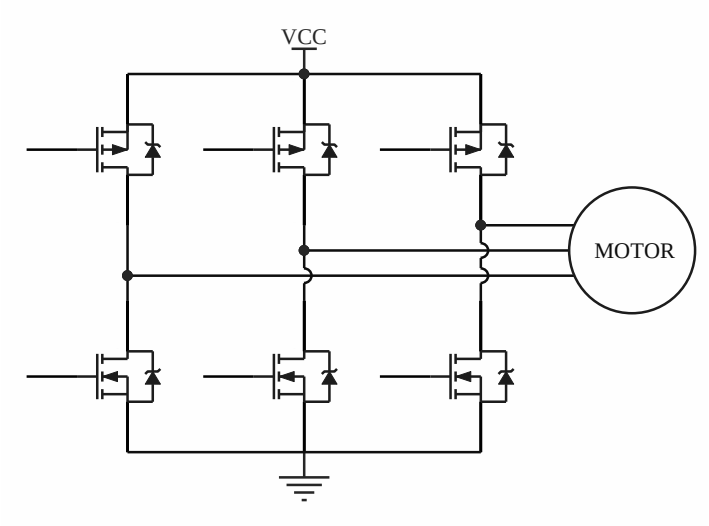


Fig. 2. The three phase inverter in complementary configuration.

3 Robot Low Level Control

The main electronics board consists of a FPGA as a single chip central controller. The FPGA is embedded with a 32-bit processor, brushless motor controller, PWM generator, quadrature decoder, kicker board controller and onboard peripheral interfacing cores: SPI and UART. The processor runs at 30MIPS as same as oscillator clock speed. We use Altium Designer and Xilinx ISE software to generate, configure and debug these cores.

3.1 Brushless Motor Driver

The three phase inverter bridge is fed with signals from FPGA to provide commutation for each motor. These signals are ANDed with the PWM signal to vary the average voltage applied to the motor winding. The six steps commutation sequence is detected by three hall sensors in the motor. Both high and low side drivers are driven by PWM signals to control the torque applied to the motor.

3.2 Motion Control

The robot employs a PI controller as a motion controller, one controller for each motor. The control loop executes 600 times per second using velocity feedback from the encoder in driving motor and hall sensors in dribbling motor. The proportional and integral gains are manually hand-tuned. The computer sends a velocity for each DOF: x-y axis and rotation axis. Then, converted to each wheel velocity and sent to

the PI controller. The output from the controller is sent directly to the PWM controller.

3.3 Over-current Protection

General problem when driving the inverter bridge is the shoot-through current. This current is caused by turning on one side of the driver immediately after the other side of the driver has been turned off, because the MOSFET turn-off time is usually higher than the turn-on time. This situation occurs when the motor is reversing direction, which can be prevented by adding a small delay time between each high and low side driver signal.

Many of robot skills use the dribbler. Some ball stealing skills can cause dribbling motor to stall when the dribbling bar is contacted with the opponent robot. The stalled motor consumes very high current and often burn the fuse out. This over-current situation can be detected by a current sensor and can be prevented by limiting a PWM duty cycle until the current drop below the safe motor operating current. Figure 3, depicts the motor stalling situation. When the motor stalled, the motor current increased and dropped in a short time due to limited duty cycle. The motor current is controlled around the threshold while the motor is stalling.

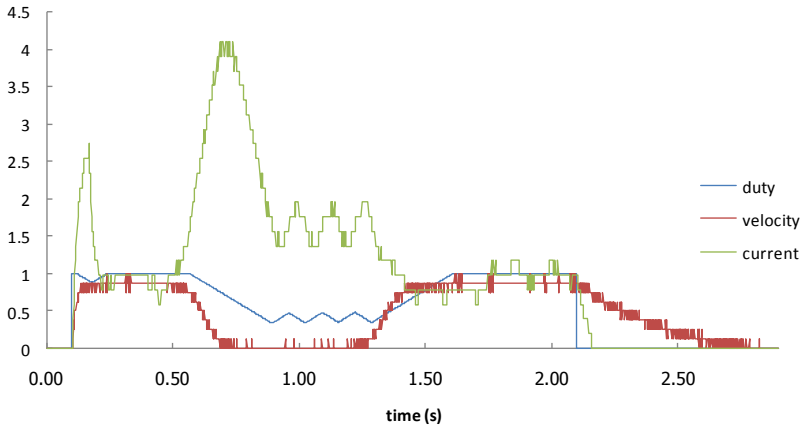


Fig. 3. PWM duty cycle, velocity and current. The motor is run from stop. Then, a very large load applied to the motor in between 0.5 sec to 1.5 sec. Current value is in ampere, duty cycle and velocity are normalized.

3.4 Torque Control for Maxon Brushless Motor

The dynamics of a robot is derived in order to provide information about its behavior. Kinematics alone is not enough to see the effect of inputs to the outputs because the robot kinematics lacks information about robot masses and inertias. The dynamic of a robot can be derived by many different methods such as Newton's law

and Lagrange equation. Newton's law is used to solve the robot dynamic equation. The interested mobile robot is consisted of four omni-directional wheels as shown in figure 4. Newton's second law is applied to robot chassis in Fig 1 and the dynamic equation can be obtained as (1) though (3).

$$\ddot{x} = \frac{\mathbf{1}}{M}(-f_1 \sin\alpha_1 - f_2 \sin\alpha_2 + f_3 \sin\alpha_3 + f_4 \sin\alpha_4) - \bar{f}_f\Big|_x \tag{1}$$

$$\ddot{y} = \frac{\mathbf{1}}{M}(f_1 \cos\alpha_1 - f_2 \cos\alpha_2 - f_3 \cos\alpha_3 + f_4 \cos\alpha_4) - \bar{f}_f\Big|_y \tag{2}$$

$$J\ddot{\theta} = d(f_1 + f_2 + f_3 + f_4) - T_{trac} \tag{3}$$

where,

$\ddot{x}$ is the robot linear acceleration along the x-axis of the global frame
$\ddot{y}$ is the robot linear acceleration along the y-axis of the global frame
M is the total robot mass
f_i is the wheel i motorized force
$\bar{f}_f$ is the friction force vector
α_i is the angle between wheel i and the robot x-axis
$\ddot{\theta}$ is the robot angular acceleration about the z-axis of the global frame
J is the robot inertia
d is the distance between wheels and the robot center
T_{trac} is the robot traction torque

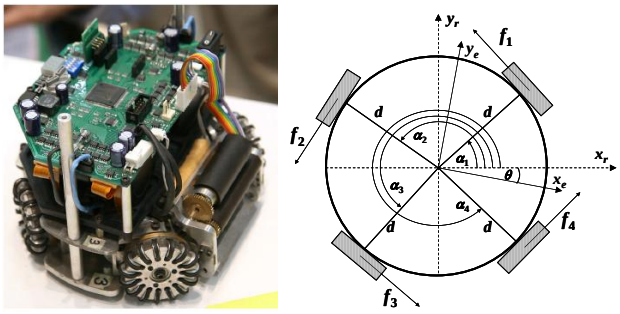


Fig 4. Robot structure

The robot inertia, friction force and traction torque are not directly found from the robot mechanic configuration. These parameters can be found by experiments. The robot inertia is constant for all different floor surfaces while the friction force and traction torque are changed according to floor surfaces. The friction force and traction torque are not necessary found at this point because these two constraints are different for different floor surfaces and their effect can be reduced by using the control scheme. The motorized forces are driven by brushless motors with τ_{m1}, τ_{m2}, τ_{m3}, and τ_{m4} therefore the dynamic equation of the robot becomes

$$\ddot{x} = \frac{1}{M}\left(-\frac{\tau_{m1}}{r}\sin\alpha_1 - \frac{\tau_{m2}}{r}\sin\alpha_2 + \frac{\tau_{m3}}{r}\sin\alpha_3 + \frac{\tau_{m4}}{r}\sin\alpha_4\right) - \bar{f}_f\Big|_x \tag{4}$$

$$\ddot{y} = \frac{1}{M}\left(\frac{\tau_{m1}}{r}\cos\alpha_1 - \frac{\tau_{m2}}{r}\cos\alpha_2 - \frac{\tau_{m3}}{r}\cos\alpha_3 + \frac{\tau_{m4}}{r}\cos\alpha_4\right) - \bar{f}_f\Big|_y \tag{5}$$

$$\ddot{\theta} = \frac{d}{J}\left(\frac{\tau_{m1}}{r} + \frac{\tau_{m2}}{r} + \frac{\tau_{m3}}{r} + \frac{\tau_{m4}}{r}\right) - T_{trac} \tag{6}$$

where,

r is a wheel radius

Equation (4) though (6) show that the dynamics of the robot can be directly controlled by using the motor torques.

A Maxon brushless motor is selected for the robot. The dynamic model of the motor can be derived by using the energy conservation law[1]. The dynamic equation for the brushless motor is

$$u \cdot \frac{\tau_m}{k_m} = \frac{\pi}{\mathbf{30,000}} \cdot \dot{\phi} \cdot \tau_m + R \cdot \left(\frac{\tau_m}{k_m}\right)^2 \tag{7}$$

where,

u is the input voltage

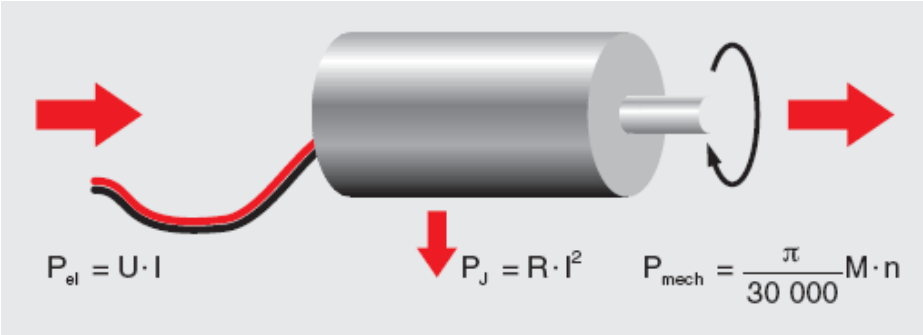
τ_m is the motor output torque

k_m is the motor torque constant

$\dot{\phi}$ is the motor angular velocity

R is the motor coil resistance

Equation (7) shows that input voltage has a linear relationship with the output torque at specific angular velocity. This equation is needed to modify to a simple version by using the Maxon parameters relationship. The final dynamic equation of the motor is derived from the following information.



Electrical power = Mechanical power + loss in wire

$$P_{el} = P_{mech} + P_{loss}$$

$$\text{electrical power} = P_{el} = V \cdot \frac{\tau_m}{k_m}$$

$$\text{Mechanical power} = P_{mech} = \frac{\pi}{\mathbf{30,000}} \cdot \dot{\phi} \cdot \tau_m$$

$$\text{Power loss} = P_{loss} = R \cdot \left(\frac{\tau_m}{k_m} \right)^2$$

$$\text{Motor constants of Maxon EC series} = k_m \cdot k_n = \frac{\mathbf{30,000}}{\pi}$$

$$\left(\frac{k_m}{\tau_m} \right)^2 \cdot U \cdot \left(\frac{\tau_m}{k_m} \right) = \left(\frac{k_m}{\tau_m} \right)^2 \cdot \frac{\pi}{30,000} \cdot \dot{\phi} \cdot \tau_m + \left(\frac{k_m}{\tau_m} \right)^2 \cdot R \cdot \left(\frac{\tau_m}{k_m} \right)^2$$

$$\left(\frac{k_m}{\tau_m} \right) \cdot U = \frac{k_m^2}{\tau_m} \cdot \frac{\pi}{30,000} \cdot \dot{\phi} + R$$

$$U = k_m \cdot \frac{\pi}{30,000} \cdot \dot{\phi} + \left(\frac{R}{k_m} \right) \cdot \tau_m$$

Use above information in equation (7), the final formula of the motor is

$$\tau_m = \left(\frac{k_m}{R} \right) \cdot u - \left(\frac{k_m}{R \cdot k_n} \right) \cdot \dot{\phi} \tag{8}$$

where,

k_n is the motor speed constant

Equation (8) shows the direct relative of the control signal *u* and the output torque τ_m at the specific angular velocity $\dot{\phi}$. The control scheme is set using the discrete Proportional-Integral control law and torque dynamic equation (8). From the experimental, the measured angular velocity has a high frequency noise therefore the low pass FIR filter is required. The error between desired angular velocity and real filtered angular velocity of each wheel is the input of the PI controller with the PI gains *kp* and *ki* respectively. The controller is shown in figure 5 and the control law can be described as (9) though (11).

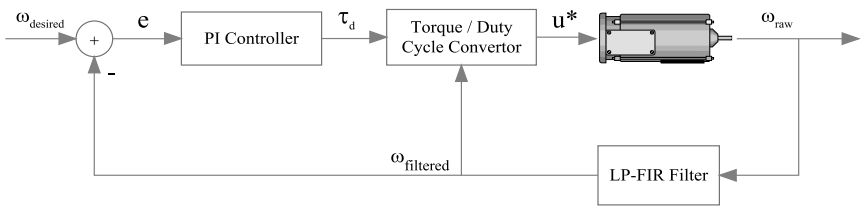


Fig 5. The Torque Controller

$$err[j] = \dot{\phi}_{desired}[j] - filtered[\dot{\phi}_{real}[j]] \tag{9}$$

$$\tau_d[j] = k_p \cdot err[j] + k_I \cdot \sum_{j=1}^{N}(err[j]) \tag{10}$$

$$u^*[j] = \frac{\tau_d[j]}{\left(\frac{k_m}{R}\right)\cdot V_{cc} - \left(\frac{k_m}{R \cdot k_n}\right)\cdot filtered[\dot{\phi}_{real}[j]]} \tag{11}$$

where,

N is the number of samples

V_{cc} is the driver supply voltage

u^* is the duty cycle of the PWM control signal

Since, the control signal for the motor is Pulse Width Modulation (PWM) signal, the output of the controller has to be the duty cycle for PWM signal generator. Equation (11) shows that the duty cycle of the control signal is the ration of the desired torque divided by the maximum torque of the motor at the particular angular velocity. The torque control loop executes 600 times per second using velocity feedback from the encoder in driving motor. The proportional and integral gains are manually hand-tuned.

To achieve the purpose of this development, the PI torque controller is implemented and applied to the wheeled mobile robot. The interested output of this experiment is the error of robot position profile when the same input velocity is applied for the different surfaces. Three different carpets are used in the experiment. The normal angular velocity control using PI controller and the torque control using PI controller are applied to the selected robot. Both controller gains are tuned until the robot can perform the same behavior on one of the test surface. This controller gains are used for all experiments. By changing the trajectory profile of the robot, the floor surfaces, the advantages of each controller are revealed. Robot and motor parameters are shown in table 1.

M	*1.5 kg*
J	*0.0192 kg/m^2*
d	*78.95 mm*
$[\alpha_1, \alpha_2, \alpha_3, \alpha_4]$	*[33,147,225,315] degree*
r	*25.4 mm*
V_{cc}	*14.8 V*
k_m / R	*0.02125* $Nm / A \cdot \Omega$
$k_m / (R \cdot k_n)$	*0.0005426* $Nm \cdot V \cdot s / A \cdot \Omega \cdot rad$

Table 1. Robot parameters

Case	$\theta(\mathbf{0})$	$\dot{x}$	$\ddot{x}$
1	90	2	3
2	90	1.5	2
3	45	0.8	1
4	0	0.8	1

Table 2. The experimental parameters

Trapezoidal trajectories for robot are generated by using the robot kinematic equation with different conditions. The experimental set-point angular velocity is shown in figure 6.

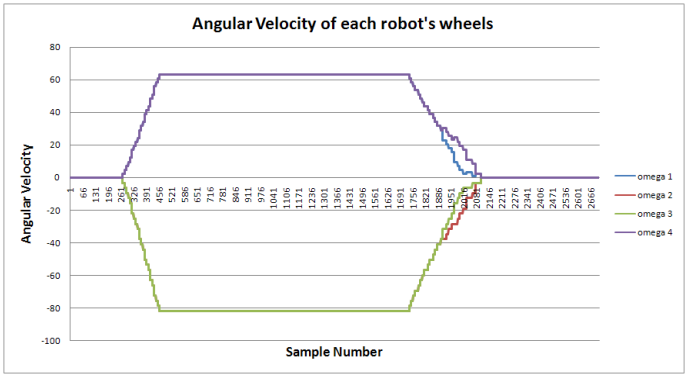


Fig 6. Angular velocity profiles for motors

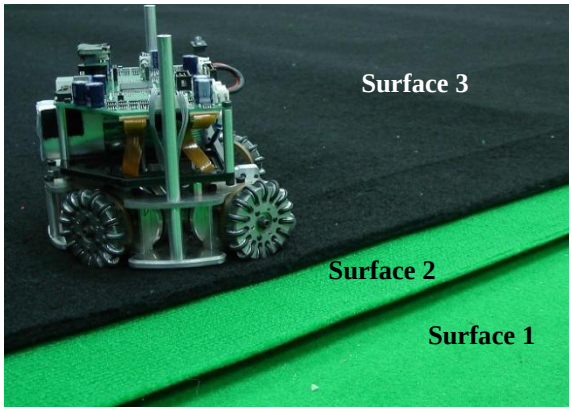


Fig 7. The selected carpet

The examples of conditions for each trajectory are shown in table 2, where $\dot{x}$ and $\ddot{x}$ are constant designed velocity of the robot along the robot's x-axis. $\theta(\mathbf{0})$ is the initial orientation of the robot. The designed trajectories are directly applied to the robot on three different surfaces without any global vision feedback controller. Figure 7 shows the selected carpets which are used in this experiment. The carpet surfaces have a different friction coefficient (μ) with $\mu_1 < \mu_2 < \mu_3$. The angular velocity of the robot

is recorded by FPGA board while the position is recorded through the bird eye view video camcorder.

The output angular velocity of motors are collected and compared with the output from same motor but different carpets. Figure 8 shows the output angular velocity of a motor which is controlled by the torque controller while figure 9 shows the output angular velocity of a motor controlled by regular velocity controller. From the experiment, a motor with torque controller can maintain its velocity when the surface frictions are changed. For a motor with normal velocity controller, the output is swing dramatically when it runs on different surface. The result shows that a motor with torque controller has a better tracking response than a motor with velocity controller especially when the motor is breaking.

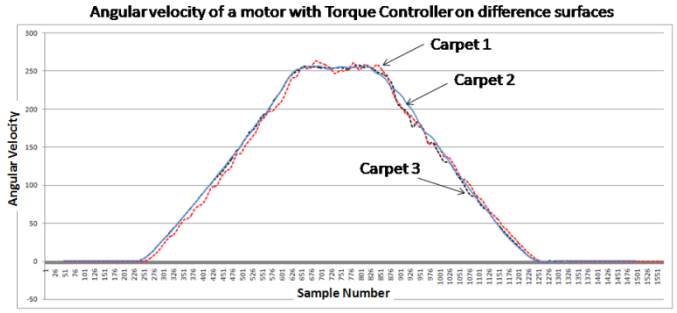


Fig 8. Angular velocity of a motor with Torque controller

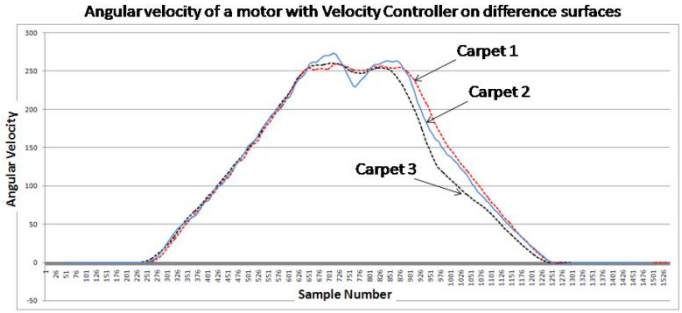


Fig 9. Angular velocity of a motor with Velocity controller

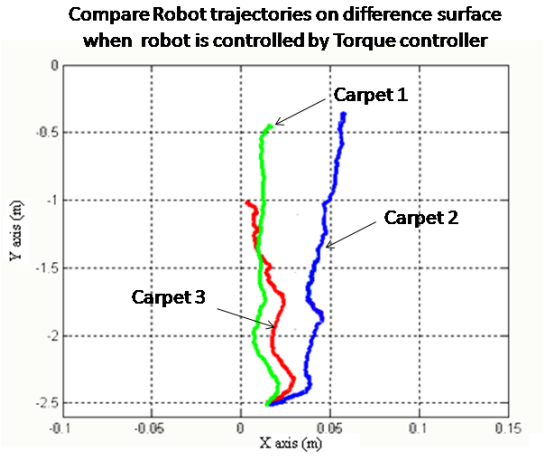


Fig 10. Position of a robot with Torque controller

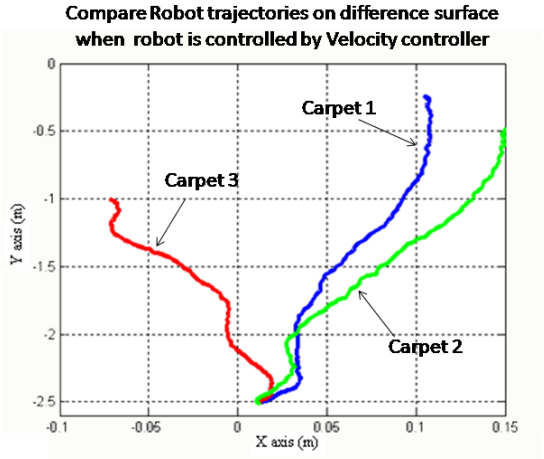


Fig 11. Position of a robot with Velocity controller

Figure 10 and 11 show the trajectory of the robot on different carpets. Robot with the torque controller has a better performance when compare with a robot with the velocity controller. Moreover, the robot path when controlled by the torque controller on the first carpet is close to the second and third carpet while the robot path when controlled by the velocity controller on the first carpet is totally difference from the second and third carpet. This result yields that if the robot turned up for one surface friction if the controller is the torque controller therefore it is possible to have a very close result when it run on the different surface frictions.

3.5 Modified Kinematic

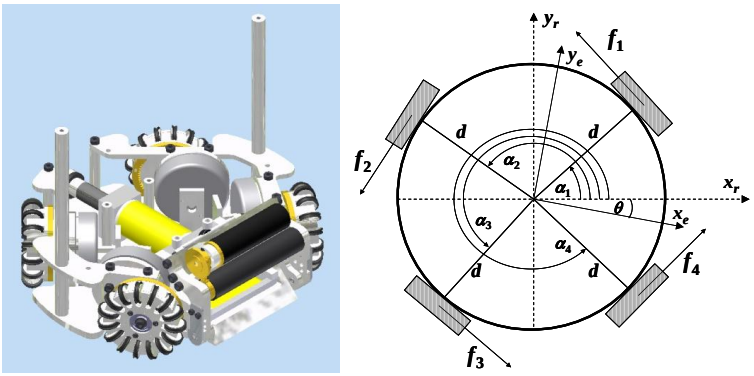


Fig 12. Robot structure

Although the robot dynamic equation can be correctly used to predict the robot behavior but it is hard to directly be implemented and it needs a long computation time. In this topic, the regular mobile robot kinematics is modified [2][3]. The normal kinematics can be written as:

$$\zeta_{earth} = (\psi^{\dagger}) \cdot \zeta_{command} \tag{12}$$

where,

$$\zeta_{earth} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix}^T$$

$$\zeta_{command} = \begin{bmatrix} \dot{\phi}_1 & \dot{\phi}_2 & \dot{\phi}_3 & \dot{\phi}_4 \end{bmatrix}^T$$

$$\psi = \begin{bmatrix} \cos\theta \cdot \sin\alpha_1 + \cos\alpha_1 \cdot \sin\theta & \sin\theta \cdot \sin\alpha_1 - \cos\alpha_1 \cdot \cos\theta & -d \\ \cos\theta \cdot \sin\alpha_2 + \cos\alpha_2 \cdot \sin\theta & \sin\theta \cdot \sin\alpha_2 - \cos\alpha_2 \cdot \cos\theta & -d \\ \cos\theta \cdot \sin\alpha_3 + \cos\alpha_3 \cdot \sin\theta & \sin\theta \cdot \sin\alpha_3 - \cos\alpha_3 \cdot \cos\theta & -d \\ \cos\theta \cdot \sin\alpha_4 + \cos\alpha_4 \cdot \sin\theta & \sin\theta \cdot \sin\alpha_4 - \cos\alpha_4 \cdot \cos\theta & -d \end{bmatrix}$$

$\psi^{\dagger}$ is the pseudo inverse of the kinematic equation

α_i is the angle between wheel i and the robot x-axis

$\ddot{\theta}$ is the robot angular acceleration about the z-axis of the global reference frame

$\dot{\phi}_i$ is the angular velocity of wheel i

d is the distance between wheels and the robot center

Equation (12) is lack of information about surface frictions. Therefore, if robot trajectories are generated from equation (12), that trajectories cannot guarantee the real robot velocity and position. Modified kinematic is introduced in order to modify the robot kinematic with friction parameters. Let the modified kinematic of the mobile robot can be described as equation (13)

$$\zeta_{earth} = (\psi^{\dagger} + \varepsilon) \cdot \zeta_{command} + \Delta \tag{13}$$

where,

ε is the Viscous friction matrix (velocity friction)

Δ is the Coulomb friction vector (surface friction)

The coulomb friction vector is easily found by experiment since this friction is constant for specific surface but the viscous friction cannot be found by simple experiment. Although the viscous friction is linearly dependent on an angular velocity of the robot chassis but this friction is a non-linear function when it's transformed to time domain. The Coulomb friction is canceled out when the torque controller is applied to robot system. Thus, term Δ is gone from equation (13). Therefore, friction matrix is a time-varying matrix which can be calculated from experiments. In order to obtain the friction matrix, the error of the robot velocity in earth frame is defined as:

$$e = \zeta_{captured} - \zeta_{earth\_idea} \tag{14}$$

where,

$\zeta_{captured}$ is a captured robot velocity by over view camera

ζ_{earth_ideal} is a ideal velocity of a robot when robot motors are motorized at command speed $\zeta_{command}$

Since the measured data always has noise therefore

$$\zeta_{captured} = \zeta_{earth} + \eta \tag{15}$$

where, η is a measurement noise

Evaluated equation (14) by putting equation (15) into (14) yield:

$$(\zeta_{captured} - \zeta_{earth\_idea}) - \eta = \varepsilon \cdot \zeta_{command} \tag{16}$$

Although, the friction matrix can be directly solved by using Moore–Penrose pseudoinverse but it is not in an equation form that can be used in advance robot parameters approximation such as Polynomial curve fitting or Kalman filter. Therefore, Kronecker product is selected to reform the equation (16). The Kronecker product is an operator that transforms a regular matrix multiply to a block matrix as:

$$\bar{A} \otimes \bar{B} = block[a_{ij}\bar{B}] \tag{17}$$

If $\bar{A}$ is $m \times n$ and $\bar{B}$ is $s \times t$, then $\bar{A} \otimes \bar{B}$ is an $ms \times nt$ matrix. Kronecker production is used to rearrange an order of $\varepsilon \cdot \zeta_{command}$. After apply the index modification, the final form of the transformation can be defined as:

Let $A \in R^{m \times l}$ and $X \in R^{l \times m}$ then

$$AX = B \Leftrightarrow (I_m \otimes X^T) \cdot \vec{a} = \vec{b} \tag{18}$$

where,

$\vec{a}, \vec{b}$ are the vector version of the matrix A, B respectively

I_m is an identity matrix with dimension $m \times m$

Equation (16) can be rearranged by Kronecker Product as the following process:

1. Let $y = (\zeta_{captured} - \zeta_{earth\_idea}) - \eta$, $x = \zeta_{command}$, $A = \varepsilon$
2. Transform $A_{mxn} x_{nx\mathbf{1}} = \vec{y}_{mx\mathbf{1}} \Leftrightarrow (I_m \otimes x^T)\vec{a}_{mnx\mathbf{1}} = \vec{y}$
3. Solve for friction matrix (now in vector form)

$$y = (I_m \otimes x^T)\vec{a}_{mnx\mathbf{1}} = \left(\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \otimes \begin{bmatrix} \dot{\phi}_{\mathbf{1}} & \dot{\phi}_{\mathbf{2}} & \dot{\phi}_{\mathbf{3}} & \dot{\phi}_{\mathbf{4}} \end{bmatrix} \right) \cdot \vec{\varepsilon}$$

where, $\vec{\varepsilon} = \begin{bmatrix} \varepsilon_{\mathbf{11}} & \varepsilon_{\mathbf{12}} & \varepsilon_{\mathbf{13}} & \varepsilon_{\mathbf{21}} & \varepsilon_{\mathbf{22}} & \varepsilon_{\mathbf{23}} & \varepsilon_{\mathbf{31}} & \varepsilon_{\mathbf{32}} & \varepsilon_{\mathbf{33}} \end{bmatrix}^T$

Finally, the friction vector estimation function is defined as:

$$\vec{\varepsilon} = (I_m \otimes \zeta_{command}{}^T)^{\dagger} \cdot (\zeta_{captured} - \zeta_{earth\_idea}) + \eta^* \tag{19}$$

where, η^* is the transformed measurement noise

ε is found from equation (19). In order to used ε , the command which is send from SKUBA system is rewritten. Since the original trajectory (command) is generated from kinematic of robot without information about friction, $\zeta_{send_{old}}$, thus, the final version of command, ζ_{send} , is described below:

$$\begin{aligned} \zeta_{send} &= (\psi^{\dagger} + \varepsilon) \cdot \zeta_{command} = (\psi^{\dagger} \zeta_{command} + \varepsilon \cdot \zeta_{command}) \\ \zeta_{send} &= \zeta_{send_{old}} + \vartheta \end{aligned} \tag{20}$$

Where,

$$\begin{aligned} \varepsilon \cdot \zeta_{command} &= \vartheta \\ \psi^{\dagger} \zeta_{command} &= \zeta_{send_{old}} \end{aligned}$$

Equation (20) now is easy to use, the friction effect is combined to matrix ϑ . This matrix can be time varying function which updated while training or constant matrix when this matrix is used during competitions. The modified kinematic is tested in 2 different surface. The result is shown in figure 13 and 14.

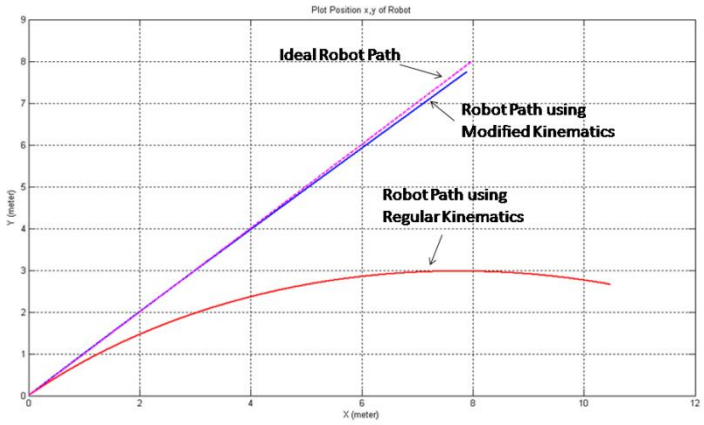


Fig 13.The result of the first surface

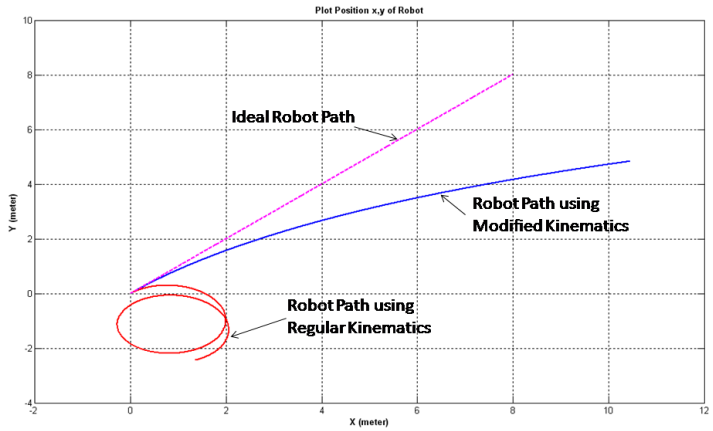


Fig 14. The result of the second surface

4 Robot High Level Control

ControlModule receives predicted vision from VisionModule and destinations from StrategyModule and makes robots go to those destinations. So, the essential component of ControlModule is a path planning algorithm. Since the World RoboCup 2008 at Suzhou[4], we have made use of the "Real-Time Randomized (RRT) Path Planning for Robot Navigation" for default path planning algorithm and use of the "Sub Goal Path Planning" for fast move planning algorithm.

After the ControlModule get command to navigate robot from the start point to the end point, the ControlModule will find the path (RRT or Sub Goal) by using start position, end position, initial velocity along x and y axis and direction of starting point and ending point. The result path will be calculated the usage time in order to use this information generates the velocity command which will be sent to the robot. Sometime the velocity is limited by the maximum acceleration of the robot. The velocity command is generated separately to the every point along the trajectory according to the frame rate. Each frame has its own velocity command. If there are any obstacles block the robot path, the path planning will be spited to small straight line to avoid collision. The velocity profile is generated by using Bang-Bang algorithm as shown in figure 15 and 16. Final motion of the robot and robot velocity profile are shown in figure 17 and 18 respectively.

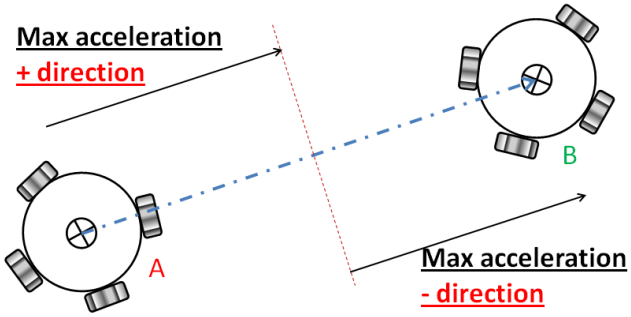


Fig 15. Bang-Bang motion

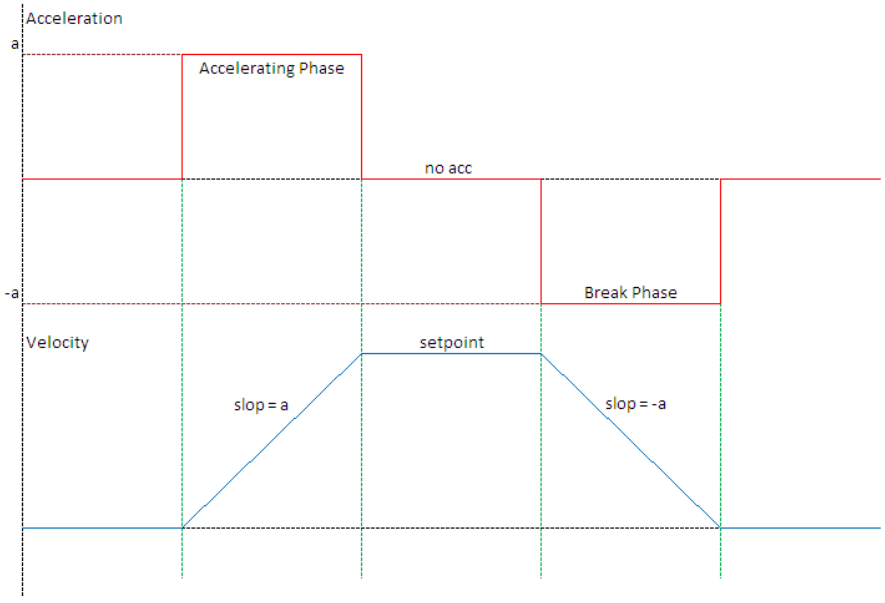


Fig 16. Bang-Bang Algorithm

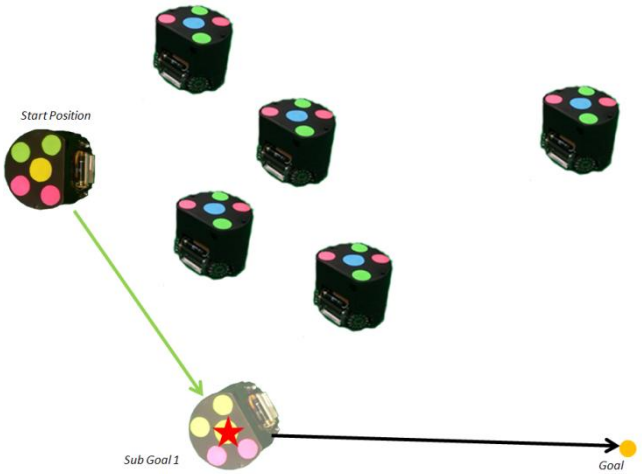


Fig 17. Robot motion and velocity profile.

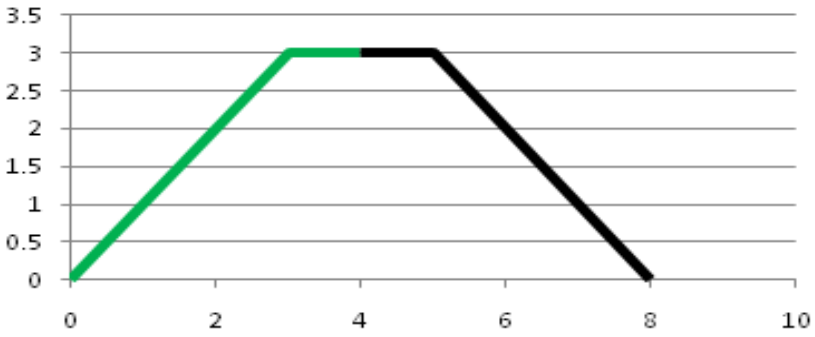


Fig 18. Robot Trapezoidal velocity profile.

The hardest calculation of the robot motion, in both RRT and Sub goal approaches, is to find the proper velocity along x and y axis according to current robot position and orientation since the possible velocity along x and y direction are not the same because of the robot structure. The robot can move with a faster velocity when it moves forward or backward and slower velocity when it moves to left or right direction. In SKUBA system based on Cornell research, the binary search is used to find the different of velocity to time. The proper velocity is separated in x and y direction.

After calculation along x and y direction, the maximum total time usage is selected in order to guarantee the robot motion. For example, if the total time usage along x-axis is greater than along y-axis, the trajectory time constrain is the total time along x-axis. The velocity that is generated for a robot in x direction will use a proper velocity along x-axis (maximum) and the velocity in y direction will use smaller value than the value that is calculated from previous step in order to make the robot move smooth.

The trajectory generation algorithm above can be written as Pseudo Code below:

Pseudo code for Trajectory_2D function

```
Trajectory_2D(StartState,FinalState,Framerate,maxAcc,maxVel,maxAngAcc,maxAngVel)
        u = PI/2
        du = -PI/2
        For i = 0 to 10 :

                alpha = u+(du *= 0.5)
                axMax = sin(alpha)*maxAcc
                ayMax = cos(alpha)*maxAcc
                vxMax = sin(alpha)*maxVel
                vyMax = cos(alpha)*maxVel

                deltaX = FinalState.X - StartState.X
                deltaY = FinalState.Y - StartState.Y

                xAcc,tx = Trajectory_1D(deltaX,StartState.Xvel,FinalState.Xvel,Framerate,axMax,vxMax)
                yAcc,ty = Trajectory_1D(deltaY,StartState.Yvel,FinalState.Yvel,Framerate,ayMax,vyMax)
                if(tx – ty <= 0.0) : u = alpha

        trajTime = MAX(tx,ty)
        deltaAng = FinalState.Rotation – StartState.Rotation
        For factor = 0.1 to 1.0:

                angAcc,tAng=
              Trajectory_1D(deltaAng,StartState.Rotation,FinalState.Rotation,Framerate,maxAngAcc*factor,maxAngVel)
                if tAng < trajTime :
                        break

        return StartState.Xvel + xAcc/Framerate, StartState.Yvel + yAcc/Framerate, StartState.Rotation + angAcc/Framerate
```

Pseudo code for Trajectory_1D function

```
Trajectory_1D(deltaS,StartVel,FinalVel,Framerate,AccMax,VelMax)
        if(deltaS = 0 and StartVel = FinalVel)
                return 0,0
        timeToFinalVel = | v0 - v1| / a_max
        distanceToFinalVel = ( | FinalVel + StartVel | / 2.0) * timeToFinalVel
        if( |StartVel| > |FinalVel| )
                timeTemp = (sqrt((StartVel^2 +FinalVel^2) / 2.0 + |x0| * a_max) - |StartVel|) / a_max
                if (timeTemp < 0.0) timeTemp = 0
                timeAcc = timeTemp
                timeDec = timeTemp + timeToFinalVel
        else if( distanceToFinalVel > |deltaS|)
                timeTemp = (sqrt(StartVel^2 + 2 * a_max * |StartVel|) - |StartVel|) / a_max
                timeAcc = timeTemp
                timeDec = 0.0
        else
                timeTemp = (sqrt((StartVel^2 +FinalVel^2) / 2.0 + |x0| * a_max) - |FinalVel|) / a_max
                if (timeTemp < 0.0) timeTemp = 0
                timeAcc = timeTemp + timeToFinalVel
                timeDec = timeTemp

        trajTime = timeAcc + timeDec
        if (timeAcc * Accmax + |StartVel| > VelMax)
                 trajTime += (VelMax - (AccMax * timeAcc + |StartVel|)) ^ 2 / (AccMax * VelMax)

        if(timeAcc < 1/Framerate and timeDec = 0)
                trajAcc = (FinalVel-StartVel)/Framerate
        else if(timeAcc < 1/Framerate and timeDec > 0)
                trajAcc = (AccMax * timeAcc) + (-AccMax * ((1/Framerate) – timeAcc))
        else
                trajAcc = AccMax
        return trajAcc,trajTime
```

"Trajectory_2D" is the function that is used to find the next command velocity that will be sent to robot next frame. The angular acceleration of the robot along x and y direction which are used in "Trajectory_2D" are found from "Trajectory_1D" function. The Trajectory_1D can be explained as:

1) If the initial velocity is less than final velocity, the robot must be accelerated until the velocity of the robot is greater than final velocity and break the robot to make the robot velocity become final velocity at the end point.
2) If the initial velocity is greater than final velocity, the robot must be decelerated until the velocity of the robot is equal to the final velocity at the end point.
3) If the distance is too small and the robot cannot be accelerated to the final velocity, the function will make the robot just accelerate without consider the velocity

After the "Trajectory_1D" function gets the acceleration or deceleration value, the usage time will be checked if it is equal to time of one frame, this acceleration or deceleration value will be used as frame acceleration but if it less than time of one frame, this acceleration or deceleration value is summered with other calculation within one frame and be averaged to averaged frame acceleration.

The "Trajectory_2D" gets the return value from "Trajectory_1D" function as frame acceleration. The "Trajectory_2D" will calculate an angular acceleration by starting small value and calculate a robot rotation usage time and compare with usage time along x and y direction (linear motion usage time) if a robot rotation usage time is greater than linear motion usage time, the current angular acceleration is increased and the calculation start again until the "Trajectory_2D" function find a robot rotation usage time less than linear motion usage time. Finally the "Trajectory_2D" function will calculate the final velocity and angular velocity that will be sent to robot as robot command. Figure 19 shows the trajectory generated from RRT and Sub Goal.

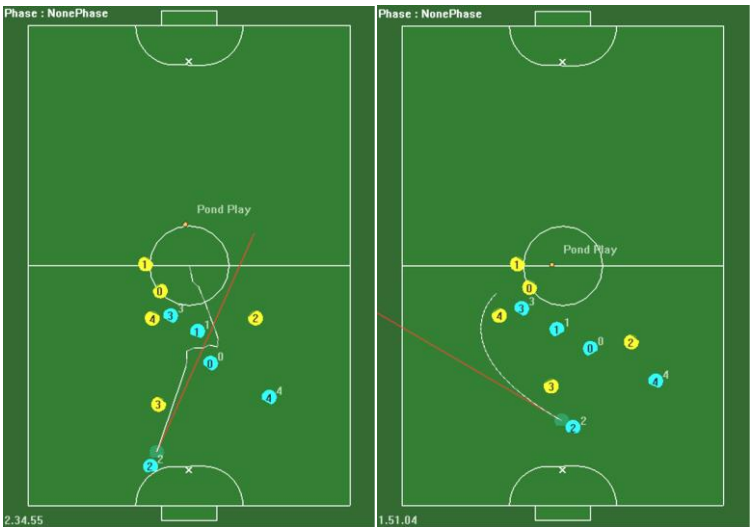


Fig 19. (Left) trajectory from RRTs, (Right) trajectory from Sub Goal.

5 Conclusion

Table 3. Competition results for Skuba SSL RoboCup team.

Competition	Result
RoboCup Thailand Championship 2005	3rd Place
RoboCup Thailand Championship 2006	Quarter Final
RoboCup 2006	Round Robin
RoboCup Thailand Championship 2007	3rd Place
RoboCup Thailand Championship 2008	2nd Place
RoboCup 2008	3rd Place
RoboCup 2009	1st Place
RoboCup China Open 2009	1st Place
RoboCup 2010	1st Place
RoboCup Iran Open 2011	1st Place

Our system has been continuously improving since the beginning. Last year, we introduced some improvements about the low level motion controller and the robot hardware. The new calibration software is fully testes in RoboCup 2010 and it greatly reducs the amount of team setup time which allowed us to focus more on the strategic planning. The software which runs the robot team was built in 2006 and improved each year. It has given us very successful competition results for the last several years, the results are summarized in table 3. We hope that our robot team will perform better in this year and we are looking forward to sharing experiences with other great teams around the world.

References

1. K. Sukvichai, P. Wasuntapichaikul and Y. Tipsuwan, "*IMPLEMENTATION OF TORQUE CONTROLLER FOR BRUSHLESS MOTORS ON THE OMNI-DIRECTIONAL WHEELED MOBILE ROBOT"*, ITC-CSCC 2010, Pattaya, Thailand, 2010, pp 19 – 22.
2. J. Srisabye, P. Wasuntapichaikul, C. Onman, K. Sukvichai, et al. "*Skuba 2009 Extended Team Description*," Proceedings CD of RoboCup 2009.
3. K. Sukvichai, P. Wechsuwanmanee, "*DEVELOPMENT OF THE MODIFIED KINEMATICS FOR A WHEELED MOBILE ROBOT*", ITC-CSCC 2010, Pattaya, Thailand, 2010, pp 88-90.
4. P. Wasuntapichaikul, J. Srisabye, C. Onman, K. Sukvichai, "*Skuba 2010 Extended Team Description of the World RoboCup 2010*", Kasetsart University, Thailand, 2010.