



College of Engineering

On Board Programming and Low Level Control of Mas-net Robots

Zhongmin Wang, YangQuan Chen

Center for Self-Organizing and Intelligent Systems (CSOIS)
Department of Electrical and Computer Engineering
College of Engineering, Utah State University
4160 Old Main Hill, Logan Utah 84322-4160, USA
Emails: zhongminwang@cc.usu.edu, yqchen@helios.ece.usu.edu
Web: <http://www.csois.usu.edu/>

December 15, 2004



Technical Report No. : [USU-CSOIS-TR-04-09](#)

Reports are available from <http://mechatronics.ece.usu.edu/reports/>

On Board Programming and Low Level Control of Mas-net Robots

Zhongmin Wang, YangQuan Chen

CSOIS, Dept. of Electrical and Computer Engineering

Utah State University

Logan, Utah 84322-4160, USA

Abstract

This report describes the onboard programming for ATMEL128 and MICA2 by using NesC language. The second part of the report talks about the low level control of the MAS-net robots.

I. HARDWARE AND PROGRAMMING TOOLS

A. *Introduction to Atmel128*

Atmel128 is the product of the Atmel Corporation [1]. It is a high-performance, Low-power CMOS 8-bit enhanced RISC micro controller. We have Atmel128L as the core processor for MICA2 [2] board. It works at 2.7 to 5.5 volts with 8MHz speed.

Our robot makes use of the following features of Atmel128[1]: 128K bytes of in-system programmable Flash, 4K bytes SRAM, 53 general purpose I/O, 32 general purpose working registers and Real Time Counter (RTC), four flexible Timer/Counters with compare modes and PWM output, 8-channel, 10-bit ADC.

Actually power constraint is a problem in our robot system. Atmel128 provides 4 different low-power consumption modes: Power-save mode; Standby mode; Extended Standby mode; Idle mode; Power-down mode. If we implement these mode while the robot is in idle state, it will helps to save power. Currently we did not implement these mode yet. One reason is, the motor consumes much more power than CPU does for the robot .

B. *Programming in NesC for Atmel128*

NesC[3] is a C-style language with object-oriented features developed at UC Berkeley. NesC is combined with TinyOS [4] OS. TinyOS also supports object-oriented features. It has many components that provide basic functions for the developer. These components(objects) usually deal with the hardware interface. This free the job of the programmer from low-level coding. Components make the coding easier, but also less efficient. TinyOS is a foreground/background operation system

in nature. It has tasks and events the CPU should execute. It is not a RTOS. TinyOS is not suitable for constructing a networked control system with low delay tolerance. It also does not provide reliable communication within a network for current hardware.

Many tools have been developed for TinyOS that can integrate the network with PC. Especially people can do data analysis in MATLAB while connected to the network directly under MATLAB.

NesC encapsulates some basic operations in C by using Macro. If programming for Atmel128 directly, we have two choices: assemble language or C. Whether programming under NesC or not, we can call the AVR lib function with C. Users may need to look into the related head file (io128.h) for Atmel128 and the AVR C library. The head file gives important description on the hardware. User may need to use the Macro defined within.

II. INTERFACE PROGRAMMING

A. *Produce PWM signals*

The extension slot on MICA2 board gives three PWM output pin: PWM0, PWM1A and PWM1B. PWM0 can output a 8 bit PWM signal from Timer/Counter 0. PWM1A and PWM1B each gives 16 bit PWM signals from Timer/Counter 1. Since 16 bit PWM gives more advantage on motor control, we assign PWM1A and PWM1B as control signal to the left and right motor respectively.

TinyOS has lots of components that free the user from the job to deal with hardware. But it does not provide component that can initiate and produce PWM signals. One reason is TinyOS is designed for sensor network, but not for sensor/actuator networks. We have to make the PWM signal work from the scratch.

One motor needs 3 control signals. Two of them are used to change the motor rotate direction or brake the motor. The other is PWM signal. One motor need another two lines for the power connection. The steps below show how to initialize the hardware for PWM output.

PWM Mode Initialization:

- Set all the control pin as output pin.
- Clear Output Compare Register.
- Set Timer/Counter1 Compare Output Mode for PWM output.
- Set Timer/Counter1 Wave Generation Mode as Fast PWM.
- Set Timer/Counter1 Control and Status Register TCCR1B to set input clock frequency.

- Set Input Capture Register (ICR1) to define the PWM period.

The duty cycle of the PWM signal is set by assigning values to Output Compare Register OCR1A and OCR1B. The ratio of $OCR1x/ICR1$ equals the duty cycle.

B. Wireless Communication

MICA2 has a low power RF transceiver, CC1000[5]. The default setting in TinyOS gives a baud rate of 19200 bit/s, which corresponds to about 50 packets/s. Every packet is maximum 34 bytes in length. The payload for each packet is maximum 29 bytes. This can be changed by modifying the AM.h head file under directory tos/types.

For general purpose, we can use the interface the GenericComm component provides. There are 3 steps for users to send a message:

- Fulfill every field in the message.
- Call Send command. User need to specify which node is the receiver. Broadcast is an option.
- When the message is sent out successfully, a SendDone event is triggered. Deal with SendDone event.

There are 3 steps to receive a incoming message.

- When a incoming message is received, a Receive event is triggered. Buffer the incoming message.
- When dealing with received messages, the first step is to make sure that this message is sent to the specified robot.
- Process the message on reception.

C. Use ADC to get sensor data

Atmel128 has 8 channel, 10 bit ADC. The ADC works in sequential manner with only one converter. The converter requires a input clock frequency between 50kHz and 200kHz. One typical conversion takes 13 ADC clocks. The first conversion after the ADC is switched on takes 25 ADC clocks. The actual sample-and-hold takes place 1.5 ADC clock cycles after a normal conversion starts and 13.5 ADC clock cycles after the first conversion starts. When a conversion is complete, the result is written to the ADC data registers [1]. This gives the constrains on the maximum sampling frequency that can be achieved.

TinyOS has ADC component to provide interface to set and access ADC channel on Atmel128. We call `ADC.getdata()` to begin a ADC conversion. When the conversion is finished, a `ADC.dataReady` event is triggered and the data is returned as the result of the conversion.

D. External Interrupt handling in NesC

MICA2 provides 4 external interrupt pins on its extension slot. But these pins and some other pins are misordered. One way to find the correct connection between pins on the extension slot and the pins on Atmel128 is to test the connection by multimeter.

Interrupt 0 and 1 is used to receive encoder pulses. The steps to set the interrupt pin is as follows:

- Set the interrupt pin as input pin.
- Set the interrupt sense control. Choose the falling edge, rising edge or low level input to generate the interrupt request.
- Enable the interrupt service.

This initialization should be done by the programmer. TinyOS provides a macro, TOSH SIGNAL, to define a interrupt service subroutine. Interrupt request is dealt with in the service subroutine.

In our implementation to detect the interruptions, we change the interrupt sense control by setting falling edge and rising edge alternatively. It helps to eliminate noise and fake pulses.

E. Operation on EEPROM

MICA2 has 4M bit EEPROM that can be used to log data. TinyOS has component to provide interfaces to access the EEPROM. To write data in the EEPROM, there are 4 steps:

- 1) Call EEPROMWrite.startWrite() to initialize the write operation.
- 2) Call EEPROMWrite.write() to write data to EEPROM.
- 3) Call EEPROMWrite.endWrite() to finish current operation.
- 4) When the operation is finished, Deal with event EEPROMWrite.writeDone and EEPROMWrite.endWriteDone();

To read data from EEPROM, there are 3 steps:

- 1) Call command EEPROMRead.read() to begin the operation.
- 2) When the reading is finished, the data is returned.
- 3) Deal with EEPROMRead.readDone() event.

Most of our experiment data were stored in EEPROM and then transmitted to base station via wireless communication.

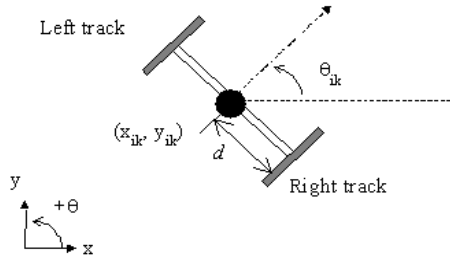


Fig. 1. Kinematic model of the 2-WMR

III. LOW LEVEL CONTROL OF WHEELED MOBILE ROBOT

A. Kinematic modelling of the two-wheeled mobile robot

To derive a kinematic model, the coordinate frame systems [6] shown in Fig. 1 are introduced. With some standard assumptions such as no friction, rigid objects, symmetric etc, the kinematic model are given as follows:

Assume that the sampling interval is ΔT , then:

$$x(k+1) = x(k) + \frac{v_l + v_r}{2} \cos(\theta(k)) \Delta T. \quad (1)$$

$$y(k+1) = y(k) + \frac{v_l + v_r}{2} \sin(\theta(k)) \Delta T. \quad (2)$$

$$\theta(k+1) = \theta(k) + \frac{(v_r - v_l)}{L} \Delta T. \quad (3)$$

where v_l and v_r are the line velocities of left wheel and right wheel, respectively. L is the distance between two wheels.

These equations are used to estimate the robot odometry. The 2-WMR has nonholonomic constraint. It has restrictions in its velocity, but these constraints do not cause restrictions in its position. We assume that the dynamic model of the motor is approximately a first order system

$$G(s) = \frac{k}{\tau s + 1}. \quad (4)$$

B. Characterization of the motor

The motor is a nonlinear system. The voltage-velocity curve for two motors on robot11 is shown in Fig. 2 and Fig. 3. The angular velocity's unit is rad/s.

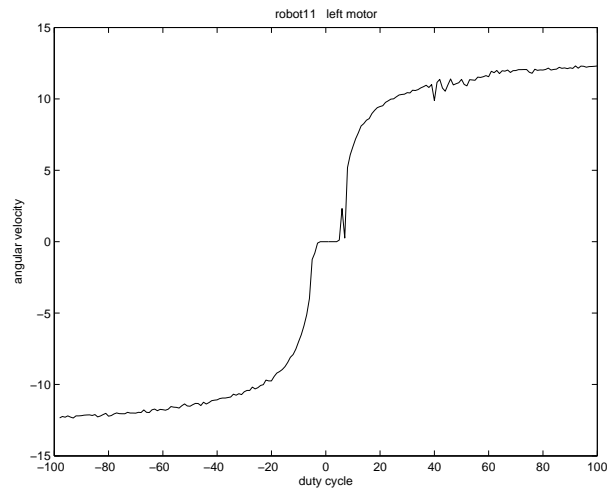


Fig. 2. Voltage (duty cycle)-angular velocity characteristic of left motor in robot 11

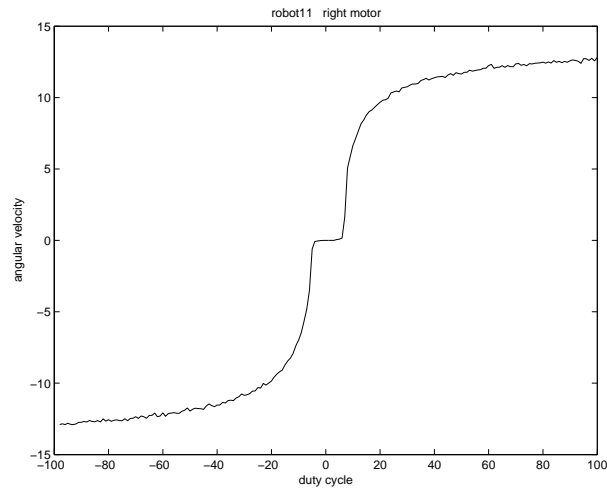


Fig. 3. Voltage (duty cycle)-angular velocity characteristic of left motor in robot 11

The experiment data is first stored in EEPROM and then transmitted to base station via wireless communication. Some data is lost in transmission.

The characteristic curve shows deadzone, saturation and other nonlinearities of the motors. The non-monotonic of the curve is caused by the friction of the floor.

The nonlinearity put some constraints on the controller design.

The time constant τ of the motor is measured from its velocity step response. τ is estimated to be 0.1s to 0.15s as can be shown in Fig. 4.

Some noise is present due to friction. The velocity is measured by

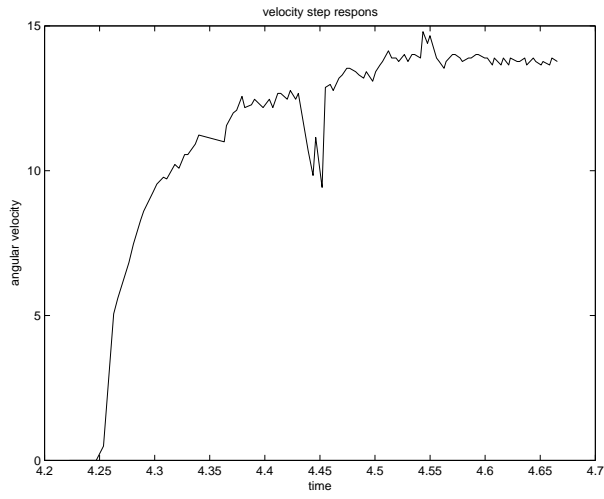


Fig. 4. Voltage step response of left motor in robot 11

$$v(t) = \frac{\Delta x}{t_{i+1} - t_i}, t \in (t_i, t_{i+1}). \quad (5)$$

Δx is the angle for each encoder sector. Ideally it is $2 * \pi / 128$ rad. $\Delta t = t_{i+1} - t_i$ is the time for the wheel to rotate one sector. When the encoder detects that one sector is passed, it will produce a pulse. The pulse is about 25 ns in time. When the CPU receives this pulse, one interruption will happen. Δt is measured by the time interval between two successive interrupts.

We can not rely on the Timer component TinyOS provides to measure the time interval. The minimal time unit for Timer component is 1 ms. This is too big for Δt . The time stamp is derived from the Atmel128 Timer/Count1 directly.

C. Open loop calibration of the motor

There are many parameters on the kinematic model of the robots. We need to know how the odometry of the robot is changed when the wheel rotates for one sector. All these data is achieved by forcing the robot to go a straight line and turn round trips.

D. Point to Point Control

The preliminary requirement on the robot control is to make it move in a line from one point to another. One simple strategy is as follows: The robot tries to head to its destination at a constant speed, as illustrated in Fig. 5

The angle between its heading and the line between the robot and the destination is the orientation error e_a . A PI controller is designed to minimize e_a . When the e_a is big, the controller output may saturate the motor. It will deteriorate

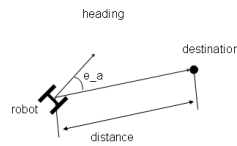


Fig. 5. The point to point movement diagram.

the system performance and cause large overshoot and slow setting time. It is called windup phenomenon. A PI controller with antiwindup [7] shown in Fig. 6 is designed.

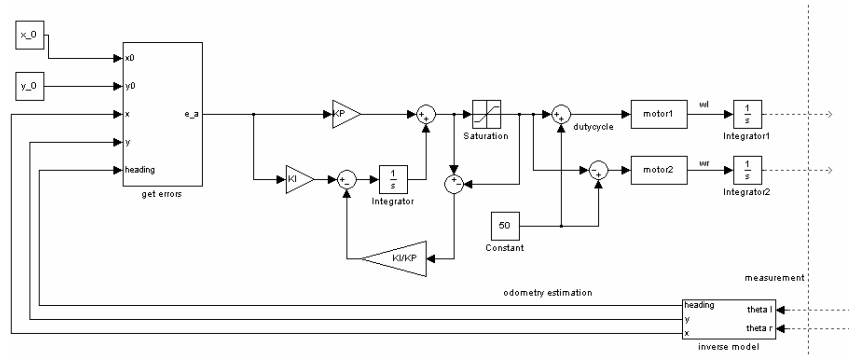


Fig. 6. The position feedback diagram

If no orientation error is present, each motors are given 50 percent dutycycle PWM signal. When the error e_a is not zero, the output of PI controller is put onto the two motor differently. For example, if e_a is greater than zero, it indicates the robot deviate from its path to its right. So, the right wheel should speed up and the left wheel should slow down to turn the robot heading towards its left, as indicated in Fig. 6. Here we assume that the angel increases in the counter-clockwise direction.

The controller parameters KP and KI is estimated in simulation and then fine tuned in the experiment. We get $KP = 80$ and $KI = 0.2$ for the best result. The heading error in the experiment is shown in Fig. 7.

The x and y position the robot estimated from its own odometry is shown in Fig. 8

E. slow start and slow stop

To achieve smooth movement for the robot, we designed the slow start phase and slow stop phase. When the robot is in start phase, the input to motor is shaped by a linearly increasing profile. When the robot is in slow stop phase, the input to motor is shaped by a profile that is proportional to the distance between robot and the destination.

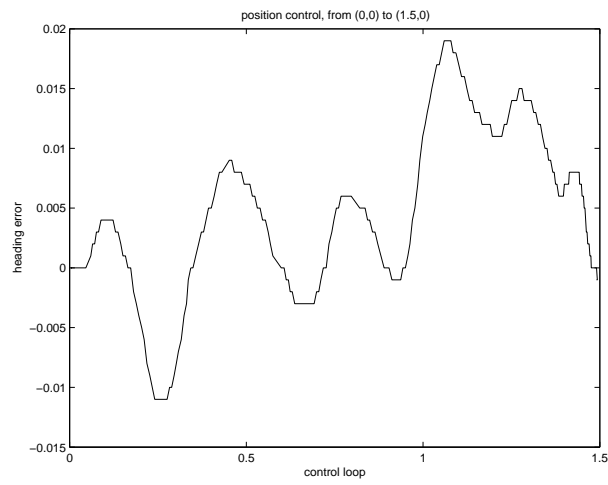


Fig. 7. The heading error for position feedback

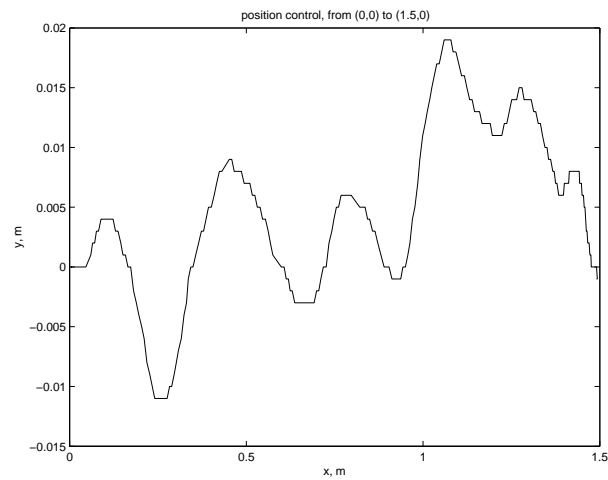


Fig. 8. Estimation of x and y from odometry for position feedback

F. PI controller for on-site turn

To make a smooth on-site turn, a PI controller is designed to control the on-site turn of the robot. One problem with on-site turn is that the wheel of the robot does not provide enough friction to make robot stop at the desired position with a medium velocity. And because of the asymmetric of the two wheels, the robot can not make an exact on-site turn.

G. Implementation in C code

The control chart is shown in Fig. 9. It uses the model of the state machine.

We have developed the Leader-Follower pattern for the robots. Please refer to the source code for details.

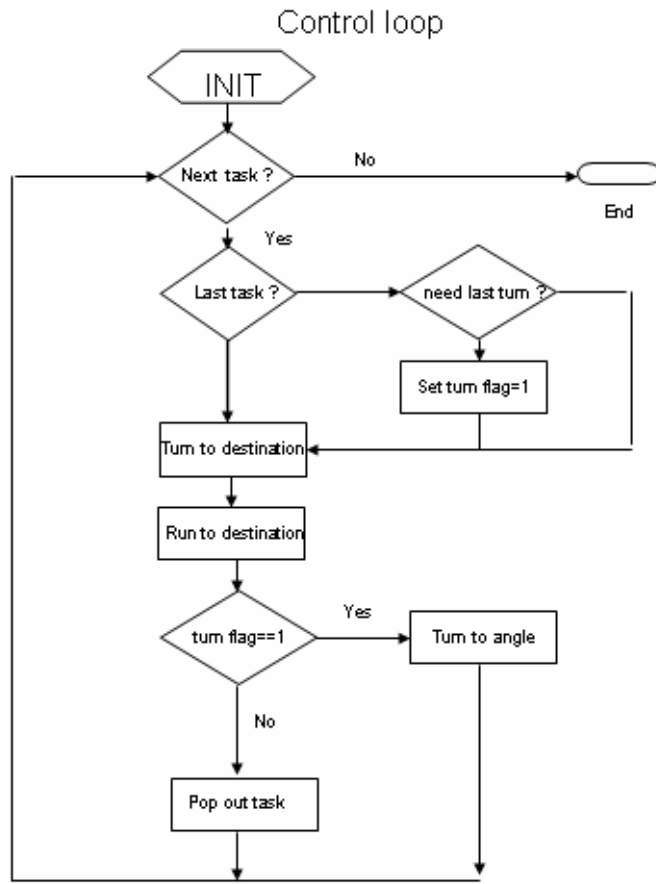


Fig. 9. Control loop chart diagram

IV. VELOCITY AND POSITION FEEDBACK CONTROL

A. velocity estimation

Velocity control is essential in rigid body formation control. There are two ways to measure the velocities.

One way is:

$$v(t) = \frac{x_{i+1} - x_i}{\Delta T}$$

Another way is:

$$v(t) = \frac{\Delta x}{t_{i+1} - t_i}$$

Here the Δx and ΔT is constant. Accordingly, we can measure the distance/angle passed in one control loop or measure the timer interval between two successively interrupts from encoder.

Experiment is done to estimate velocity by method 1 and 2. The motor is driven by 30 percent duty cycle. For method 1, we get the angle data per 20ms. The angle data is shown in Fig. 10.

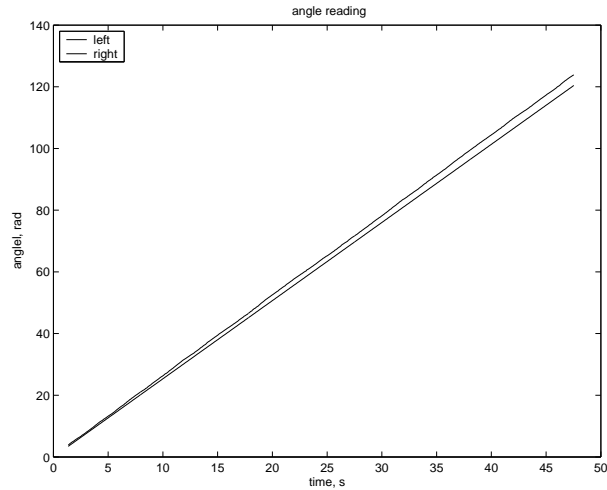


Fig. 10. Angle measurement, sample time = 20ms

After 1-order differentiation, the velocity is shown in Fig. 11. The noise caused by quantization error can be seen.

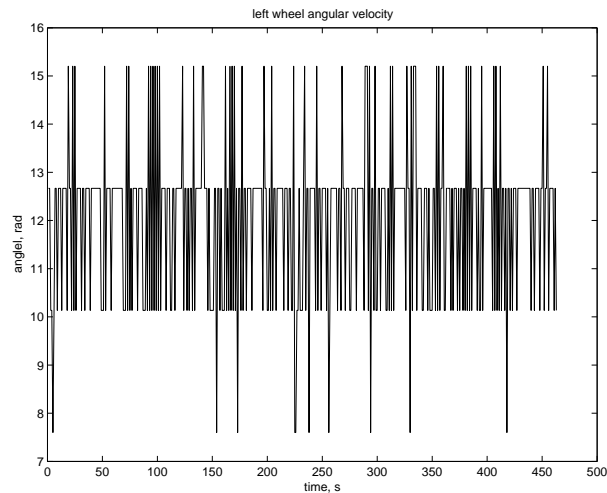


Fig. 11. Velocity measurement by method 1, sample time = 20ms

For method 2, the result is shown in Fig. 12.

A close look at the graph shows a 4-point pattern in the velocity measurement, as can be seen in Fig. 13

Based on this observation, we filter the data by averaging them by 4. The result is shown in Fig. 14.

Fig. 15 shows the output of a low Butterworth filter and the averaging method. The Butterworth filter has cutoff frequency at $0.2 \cdot \pi$. It shows that averaging method is better. If averaged, the velocity signal gives variance $2.1036 \text{ rad}^2/\text{s}^2$.

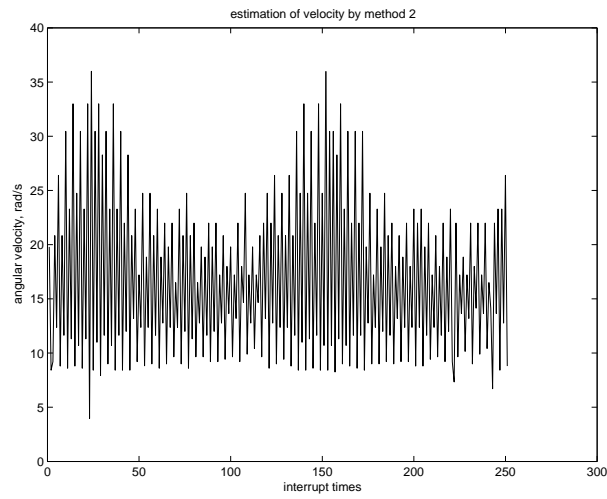


Fig. 12. Velocity estimation by method 2

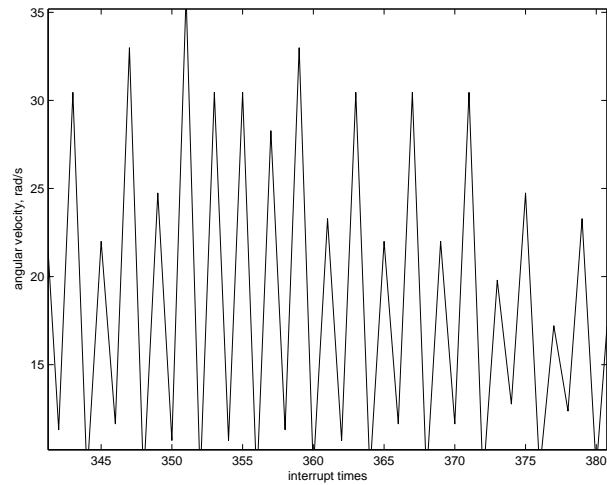


Fig. 13. Velocity estimation by method 2. Show 4 point pattern

B. Noise analysis of velocity measurement by method 2

We did some experiments to sample the velocity signal every 3 ms. The result goes through FFT analysis and is plotted in Fig. 16.

It can be seen that 3 major harmonics exist in its frequency spectrum. The lowest one is estimated to have a period about 0.46s It is the time robot takes to have one rotation. This noise is made by eccentricity of wheel axis from motor axis. The second one has a period of 14ms. This corresponds to 4-point pattern we have known. The highest frequency has period of 7.2 ms. This corresponds to a 2-point pattern.

These patterns are caused by misalignment of the sensors and the codewheel sticker [8].

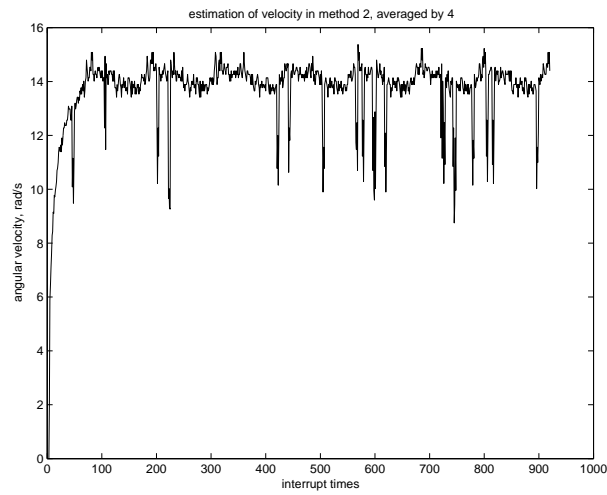


Fig. 14. Velocity estimation by method 2 and averaging by 4

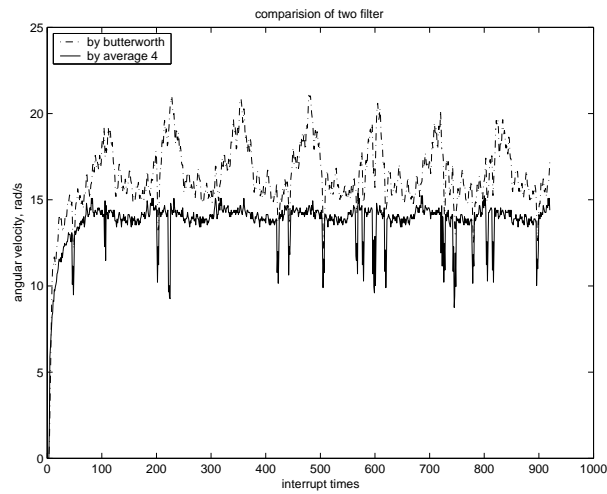


Fig. 15. Compare velocity estimation by butterworth and averaging filters

C. Velocity feedback control of robots

We use the second method to measure velocity. The velocity feedback controller employs high gain P controller and an I controller. To protect from motor saturation, antiwindup is considered, as shown in Fig. 17

The P and I controller parameters are derived and then fine tuned in experiment. The parameter for P controller is set to be 150 and the parameter for I controller is set to be 0.5. The P controller can not be set too big because the quantization error would produce limit cycles. The limit cycles will result in high frequency oscillation in the motor. To mitigate the negative effect of nonlinearity of the motor on the whole system, the static linearization method can be used.

In one experiment, the velocity set point is 0.28m/s. The velocity of left wheel, right wheel and corresponding velocity

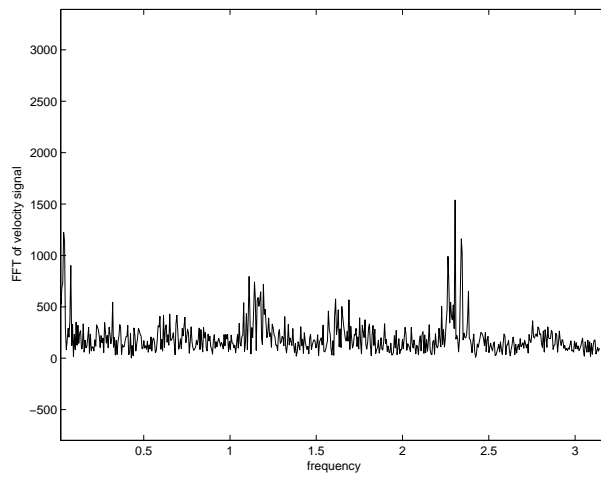


Fig. 16. FFT of velocity signal

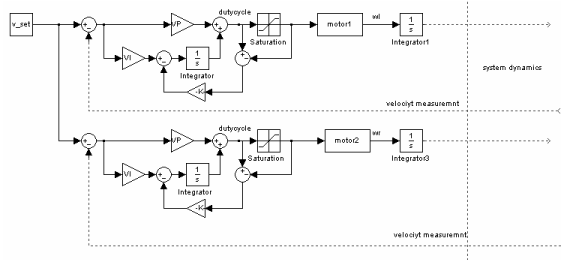


Fig. 17. Velocity feedback diagram

errors are shown in Fig. 18, Fig. 19, Fig. 20 and Fig. 21, respectively. The left wheel velocity gives variance of $0.0017 \text{ rad}^2/\text{s}^2$.

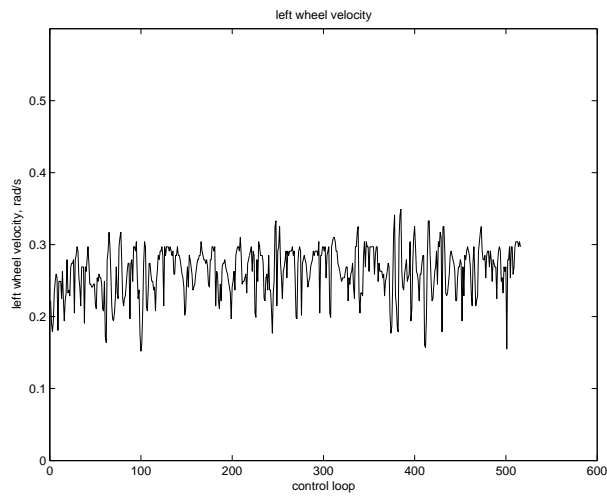


Fig. 18. Left wheel velocity

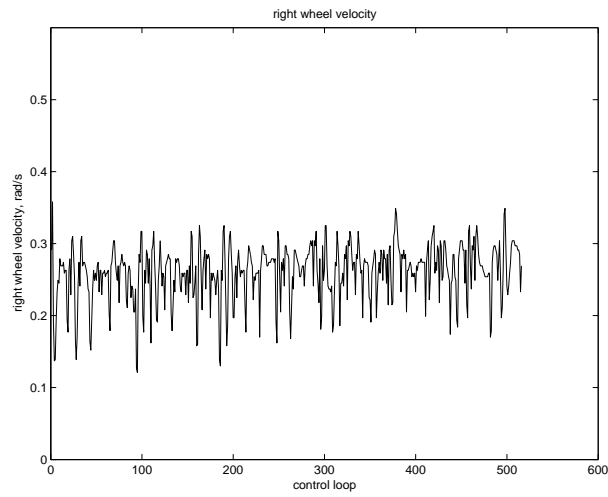


Fig. 19. Right wheel velocity

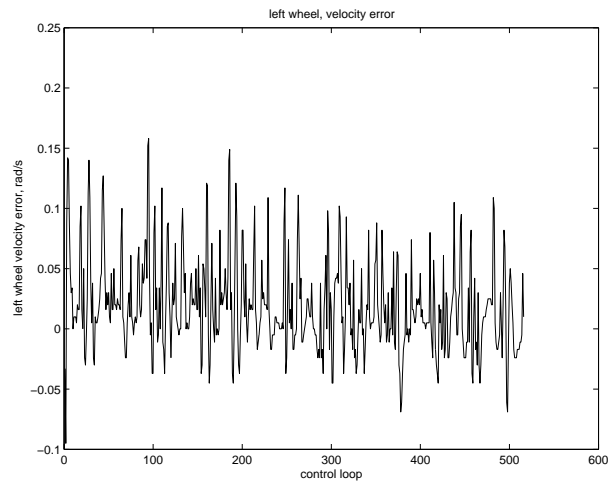


Fig. 20. Left wheel velocity error

D. cascade-loop controller

Velocity and position feedback differs from pure position feedback in that a velocity profile is designed for the robot movement. Velocity control loop is the inner loop and the position feedback loop is the outer loop. The principle of outer loop is the same as described above. The cascade loop system is show in Fig. 22.

Experiment has been done on the position/feedback control of the robot. In this experiment, the robot will stop when it finds itself within 1cm from the destination. The velocity is set to 0.35s/m. The P and I controller for outer/position loop are set to 5 and 0.01, respectively. The P and I controller for inner/velocity loop are set to 150 and 0.6, respectively. The robot stop at the destination successfully. The heading error is shown in Fig. 23.

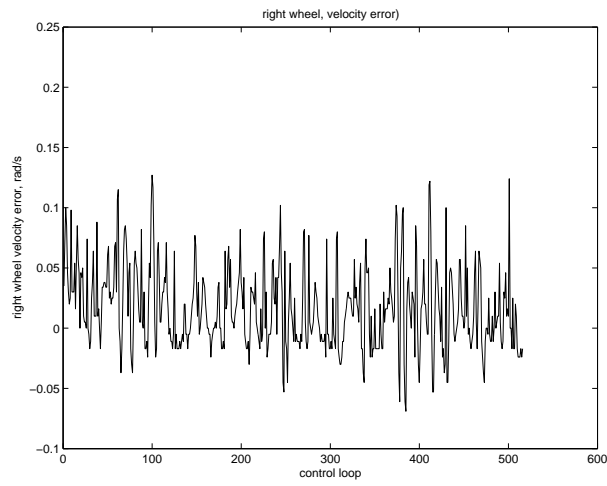


Fig. 21. Right velocity error

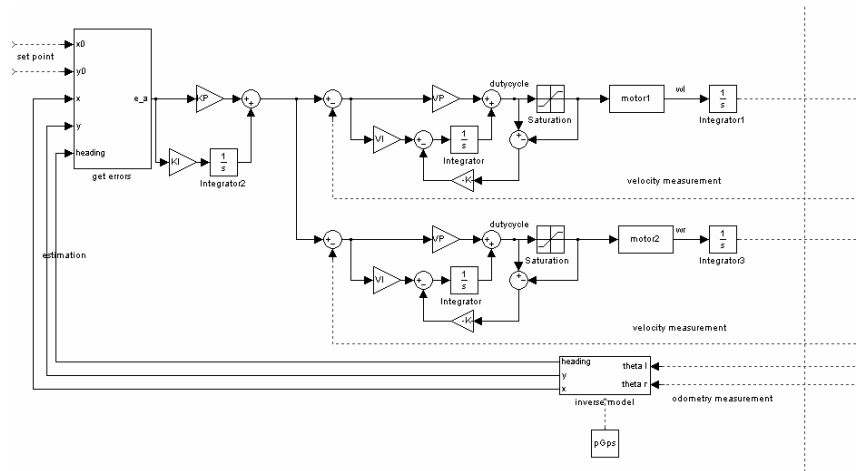


Fig. 22. Position and velocity feedback diagram

V. REACTIVE BEHAVIOR OF ROBOTS

The Mas-net robot has two IR sensors on its front that can be used to implement obstacle avoidance for the robot.

One of the typical IR characteristic curve is shown below. The output reading of 1024 corresponds to V_{cc} in CPU. Some pre-filtering is needed because of measurement noise.

It can be seen that if the IR threshold is set to about 400, the robot can find that it faces an obstacle about 15 cm ahead. This is a reasonable distance for the robot to take some actions to avoid the obstacle. The size of the robot is less than 8 by 8 cm, so it has enough room to make a on-site turn. The avoidance strategy is very simple.

- If the left IR reports obstacle, the robot turns to its right.

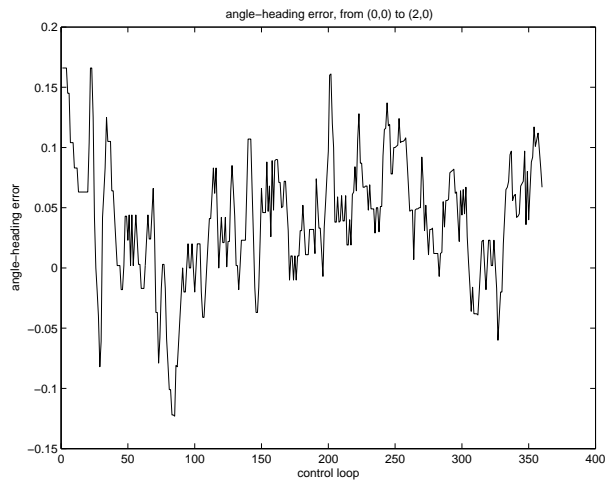


Fig. 23. Heading error in position and velocity feedback control

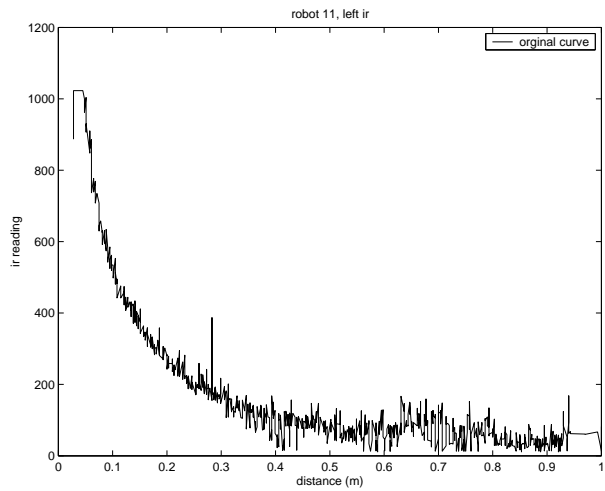


Fig. 24. IR characteristic curve

- If the right IR reports obstacle or both the left IR and right IR report obstacles , the robot turns to its left.
- When no obstacle is present after its turn, the robot tries to resume its straight line movement.

The robot also can be programmed to have other exception-dependent behavior. For example, the robot will stop itself if it finds that it is out of the specified boundary. In mobile sensor network, power constrain is a big issue. The robot can measure its voltage on digital circuit. If it is too low, the robot can stop itself and send exception message to the base station.

VI. REFERENCES

[1] ATMEL, *Atmel128(L)*, Atmel Corporation, rev. 2467gavr09/02 edition.

- [2] Crossbow, *MPR/MIB Mote User Manual*, Crossbow Technology, Inc, 41 Daggett Dr, San Jose, CA, b edition, May 2003.
- [3] David Gay, Philip Levis, David Culler, and Eric Brewer, *NesC 1.1 Language Reference Manual*, May 2003.
- [4] “TinyOS,” <http://webs.cs.berkeley.edu/tos/>.
- [5] Chipcon, *SmartRF. CC1000 PRELIMINARY Datasheet*, rev. 2.1 edition, April 2002.
- [6] “The MilliBots Project,” <http://www-2.cs.cmu.edu/~cyberscout/localization.html#system>.
- [7] Hwi-Beom Shin, “New Antiwindup PI Controller for Variable-Speed Motor Drivers,” *IEEE Trans. of Industrial Electronics*, vol. 45, no. 3, pp. 445–450, Jun 1998.
- [8] “WheelWatcher Frequently Asked Questions,” <http://www.nubotics.com/support/index.html>.