JOYSTICK CONTROL FOR TINYOS ROBOT

by

Marga Chiri, Sarah Bergbreiter and Kris Pister

Memorandum No. UCB/ERL M02/24

8 August 2002

JOYSTICK CONTROL FOR TINYOS ROBOT

by

Marga Chiri, Sarah Bergbreiter and Kris Pister

Memorandum No. UCB/ERL M02/24

31 December 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Joystick Control for TinyOS Robot

Marga Chiri, Sarah Bergbreiter, Kris Pister
Summer Undergraduate Program In Engineering
Research at Berkeley (SUPERB 2002)

Department of Electrical Engineering and Computer Science
Berkeley Sensor and Actuator Center
University of California, Berkeley

Joystick Control for TinyOS Robot

Marga Chiri

Faculty Mentor: Kristofer Pister

Graduate Mentor: Sarah Bergbreiter

August 8, 2002

Abstract

The goal of this project was to write a program that controls a TinyOS robot with an off-the-shelf joystick. To accomplish this goal, we created a MATLAB program that reads position information from a joystick, builds a TinyOS packet, and sends that information to the serial port. We connected a mica mote, running a standard TinyOS generic base station program to the serial port, which relays the TinyOS packet to a mobile robot via the RF stack. The mobile robot, running a TinyOS application program, then processes this packet and drives according to the Joystick. Thus, we were able to drive our TinyOS robot remotely.

1. Introduction

There are many robot applications for which Joystick control is useful. For instance, the NEST (Networked Embedded Systems Technology) team at UC Berkeley is currently working on developing aerial vehicles and ground vehicles to act as pursuer robots, which try to capture evader robots within a given bounded area. The pursuers are designed to operate autonomously while a human controls the evader from a distance [3]. This pursuit-evasion project will use a MATLAB interface to send a packet from a joystick or a keyboard to drive the evader. Joystick control is also useful for a fire fighter to remotely control robot vehicles to fight a fire, and for a health care giver to drive a robot around the user's room.

1.1 The Robot

The robot is a modified Kyosho Mini-Z RC toy car with attached mica mote (see Figure 1). The hardware of the Mini-Z RC car has the capabilities of setting speed, turn and direction. A Motor/Servo printed circuit board has been designed to connect through Mica's data bus. The board uses two MOSFET H-bridges to convert the control signal from the Mica mote to signals for the motors. This board can control up to two motors, servos, solenoids or a combination of these [5]. The H-bridge circuit and mica mote are described below (see section 1.1.1 & 1.1.2).

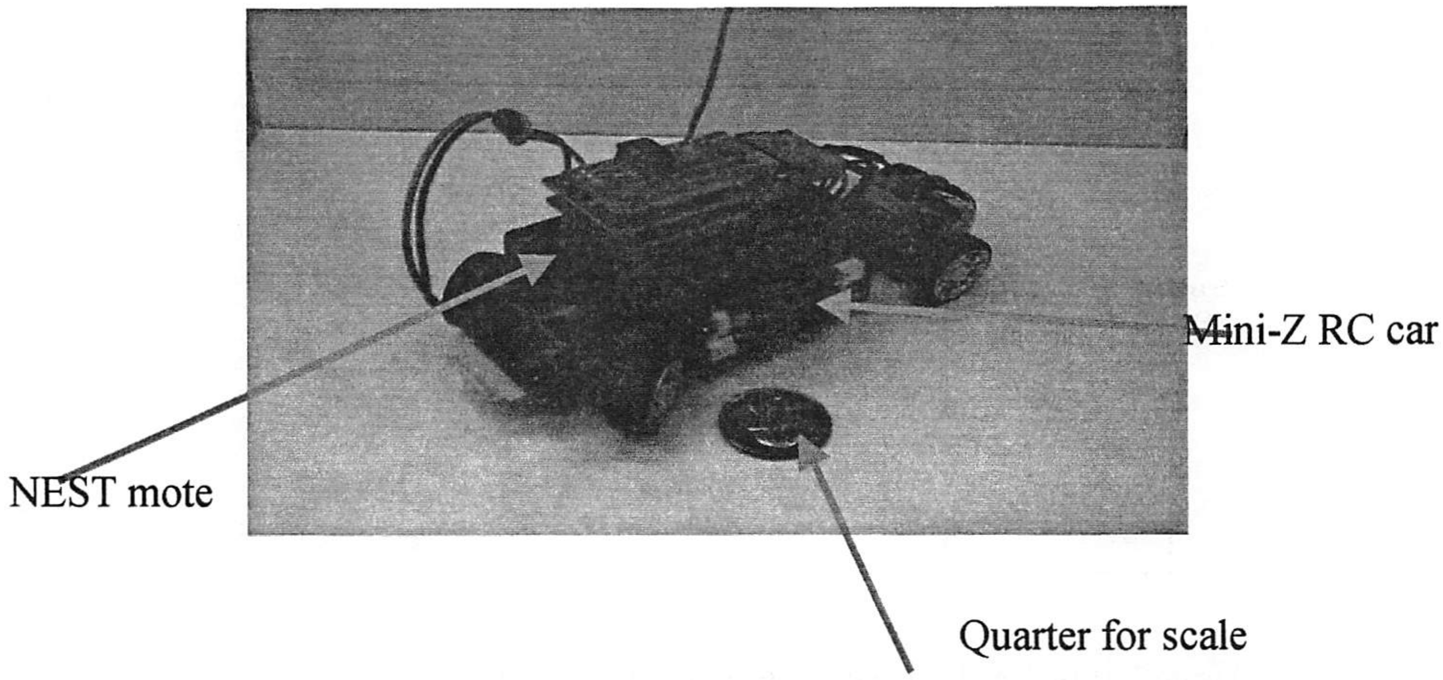


Figure 1: The Robot

1.1.1 H-bridge Circuit

The robot uses two H-bridge circuits, each of which have four transistors on its legs and a motor on the crossbar (see Figure 2). An H-Bridge circuit uses pulse-width modulation, or turning switches in the H-bridges on and off for various lengths of time, to control the speed of the motor. The motor will have a voltage equal to the supply voltage when S1 and S4 are closed and 0 V when they are open. Direction depends on the current flow across the motor. Current flows from left to right if transistors S1 and S4 closed together at the same time. If transistors S2 and S3 are closed, current flows through the motor from right to left [2].

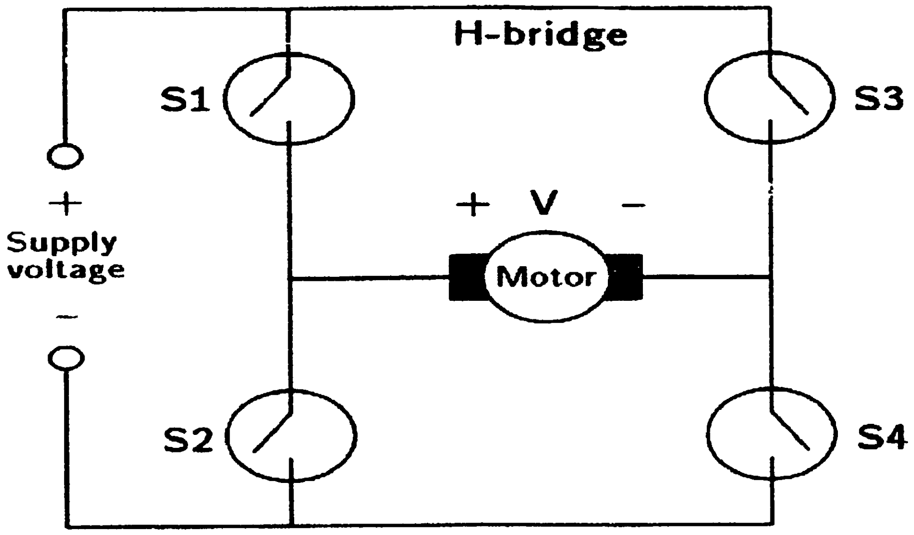


Figure 2: H-Bridge Motor Driver Circuit (Diagram from [2])

1.1.2 Mica Mote

The project used two mica motes (see Figure 3) along with a mica programming board to handle information flows. The mica mote has a microcontroller with internal flash program memory, data SRAM and data EEPROM. It also includes an 8 channel, 10-bit Analog to Digital converter (ADC). These are connected to a low-power radio transceiver, and a serial port [1]. The mica mote is programmed using TinyOS (section 1.3).

The transmission of all messages is done through the RF radio of the mote. The radio consists of an RF Monolithics 916.50 MHz transceiver (TR1000), and antenna. The radio has a short communication radius of approximately 10 meters and operates at communication rates up to 115Kbps [5].

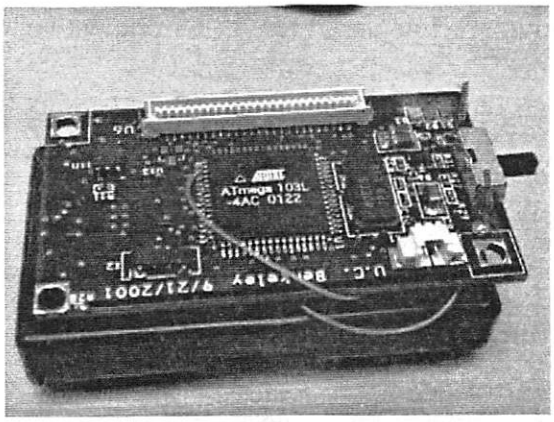


Figure 2: The Mica mote

1.2 Tiny OS

Tiny OS is a small event-driven operating system that was created by David Culler's group in the EECS department at UC Berkeley [1]. The implementation language for the system is similar to C. The complete system configuration consists of a scheduler and a graph of components. The components are composed of command handlers, a set of event handlers, variables, and simple tasks. Commands post a task to a lower level component. Event handlers generally deal with hardware. Tasks are provided to do the main computation of each component.

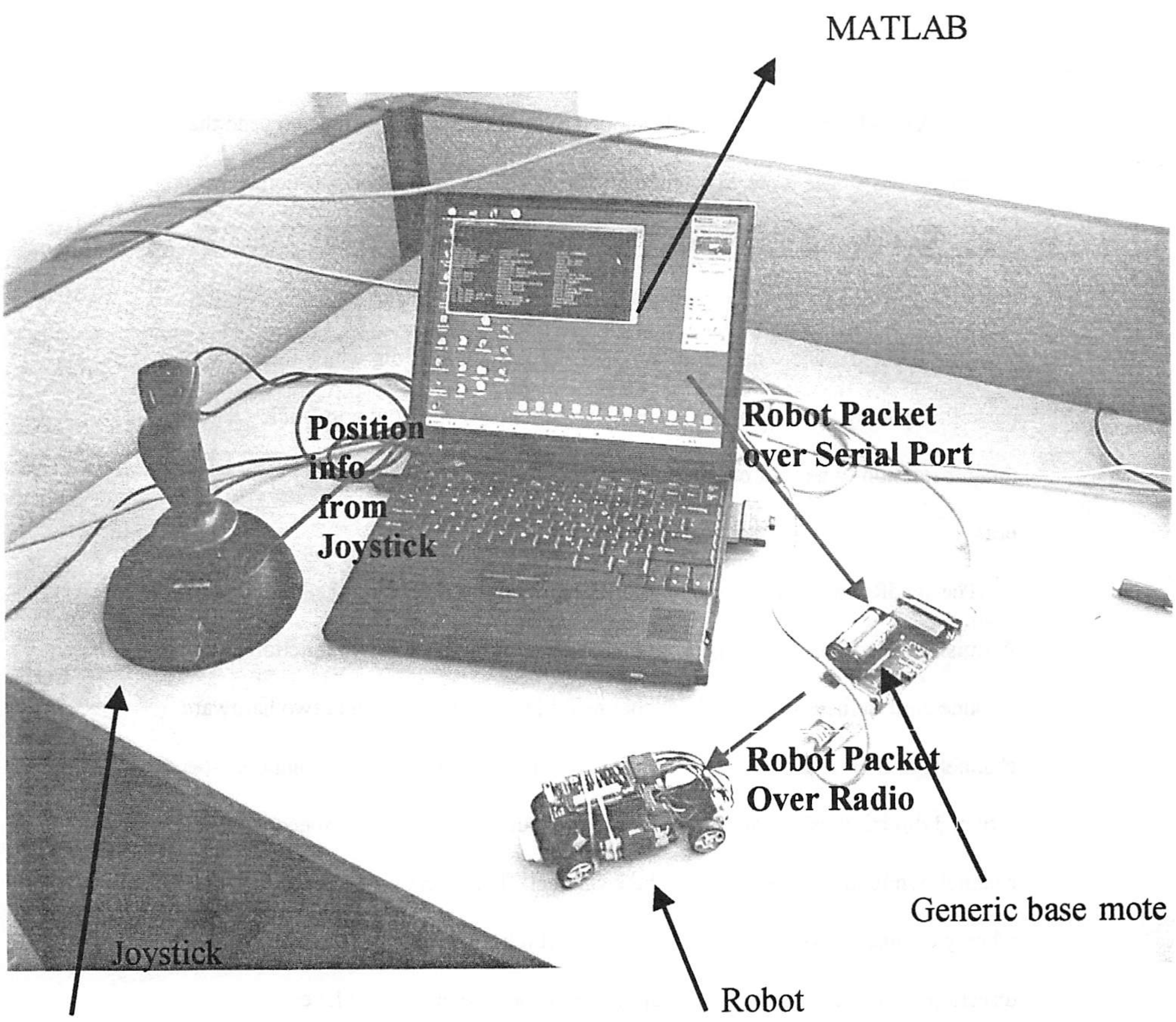
The TinyOS communication model has a command for initiating a message transmission, and signals an event on the completion of transmission or the arrival of a message. Messages in Tiny OS follow the Active Message (AM) model so that each message or packet contains a destination address, a handler number, a data payload, and a CRC checksum.

1.3 MATLAB

The NEST group at UC Berkeley has also written a MATLAB interface, which provides an easy way to prototype a TinyOS application and to analyze data. MATLAB has an interactive, interpreted environment and powerful data analysis tools, and is therefore used in many TinyOS applications. This project used the MATLAB environment to retrieve data from the joystick and to route a data packet to the robot.

2. Problem Statement

The primary objective of this project was to drive a robot using an off-the-shelf joystick. Our first goal was to write a MATLAB program, which could read data from the joystick through the USB port, convert that information to a TinyOS packet structure and send that information to a base mote attached to a PC via the serial port. A generic base TinyOS program forwards a packet from the serial port to the radio on the mica mote. Finally, we wrote a TinyOS program to receive the packet and drive a robot (see Figure 4).



gure 4: Experimental setup

3. Implementation

3.1 The MATLAB Code

To use the MATLAB environment, we need to first define a TinyOS environment by running "defineTOSenvironment.m" script on "nest/tools/matlab" directory. This defines things like variables global to our packet structure and TinyOS packet structure. We then change our directory to matlab/contrib/robotControl and run yourWorld.m script to define a communication stack for the local serial port [4].

A MATLAB application was developed to read from the joystick and send that information to a base mote. To accomplish this goal, a robot packet structure with a destination address of 65535 (the standard broadcast address), an AM handler of 8, a groupID of 125, a data payload of speed, turn and direction, and a CRC checksum was first defined. The sendRobotPacket MATLAB application, which takes the packet frequency and maximum speed as arguments, reads from a joystick, changes the information to a robot packet structure and sends that packet to the local serial port.

The sendRobotPacket program interfaces to the joystick using the MATLAB Data Acquisition Toolbox [7]. This program used analoginput ('joy', 1) function to create a connection to the joystick and addchannel(ai,[1 2]) function to add two hardware channels, one for x and one for y coordinates. This application then computes speed, turn and direction using the value of x and y channel. The value of speed is the y channel, while turning depends on the x channel. The direction becomes forward when y channel become positive and backward otherwise. Once the speed, turn and direction have been calculated, the application uses the messaging layer routePackets(2,robotPacket) to route the packet to the serial port.

3.2 The TinyOS Code

3.2.1 GENERIC BASE: forwarding a message

When the MATLAB application sends a message to the serial port, the GENERIC BASE Tiny OS application forwards all incoming UART messages to the radio. This application is included in the standard TinyOS release.

3.2.2 ROBOTCOMMAND: receiving a message

The ROBOTCOMMAND application uses the GENERIC_COMM stack, which receives the message when a packet at arrives the radio of the mica mote from the generic base station. After ROBOTCOMMAND receives the packet, it uses the lower level component MZ to set the speed, turn and direction (see Figure 5 for a graphical view).

3.2.3 MZ

Once the MZ Tiny OS component gets the speed, turn and direction, it uses the Motor1 component to set the speed and direction, and the MZ_SERVO component to set the turning angle of the Mini-Z RC robot car. These components are described below.

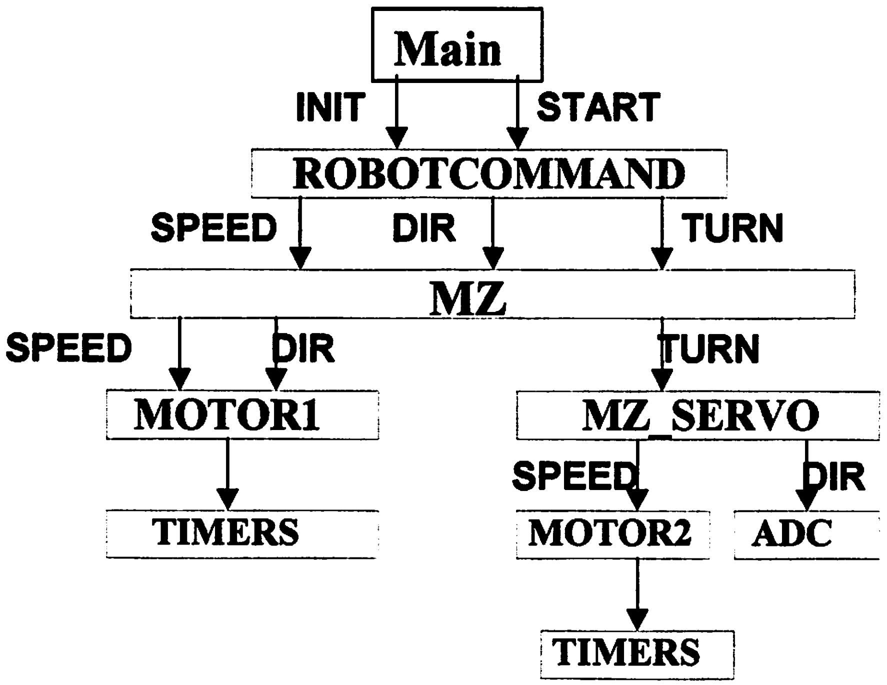


Figure 5: A graphical description of ROBOTCOMMAND TinyOS application

3.2.4 MOTOR1

The MOTOR1 component gets speed, and direction from MZ and calculates a control signal to control the speed and the direction of motor1. This component takes a speed from 0 to 10 and either forward or reverse direction. MOTOR1 uses pulse width modulation (PWM) or duty cycle modulation to determine the fractional amount of full power delivered to the motor of our robot, which controls the speed. The PWM controls the H-Bridges mentioned in Section 1.1.1.

To create the PWM signal, the MOTOR1 component uses Timers component, which fires an event every 1/1000 sec. MOTOR1 then uses an interval counter to set the speed from 0-10. This counter resets every 10 timer events, which gives PWM signal with frequency of 100Hz. To turn on the motor of our robot with half of full power in the forward direction, the MOTOR1 component takes a speed 5 and sets the robot motor PWM pin (closes a pair of diagonally opposite switches on the H-bridge) for 5 ticks (as seen in Figure 6) and then clears the motor PWM pin (open a pair of diagonally opposite switches on the H-bridge) for the remaining 5 ticks. This is shown graphically in Figure 6.

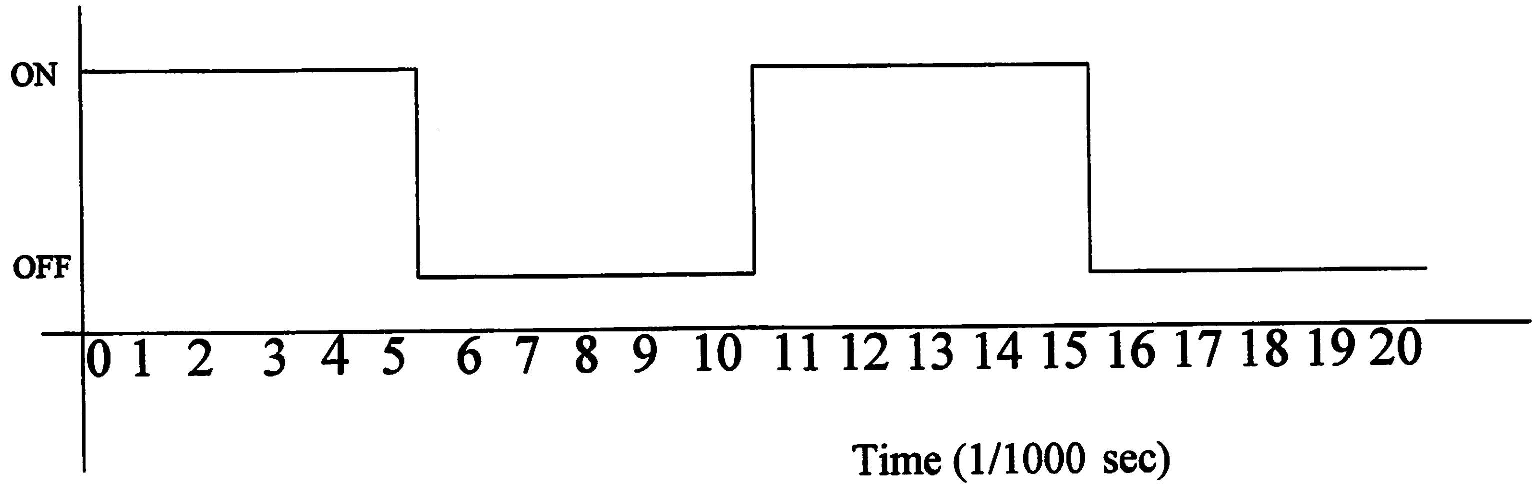


Figure 6: A 50% duty cycle for MOTOR1 component

3.3.5 MZ_SERVO

The MZ_SERVO component uses the Analog to Digital Converter on the mica mote to get position information from the potentiometer on the Mini-Z servomotor and MOTOR2 to change turning angles. A Proportional Integral Derivative (PID) control loop moves the motor to the correct position. For example, suppose the MZ_SERVO component received a command to turn 45 degrees left. If the robot turned 30 degrees left, the difference between the command input and the feedback input would generate an error signal. The PID control loop in the MZ_SERVO manipulates the error by proportional constant to produce a new command to turn the robot to the correct position.

3.3.6 MOTOR2

The MOTOR2 component is the same as MOTOR1, but uses different pins on the microcontroller.

4. Conclusion

In conclusion, we have written an interface from an off-the-shelf joystick to a TinyOS robot. This interface consisted of three separate parts: Matlab, Generic-Base and RobotCommand. The MATLAB reads from the joystick, changes the information to a TinyOS packet and sends the packet to a serial port. The Generic-Base sends the packet from the serial port to a mobile robot via the RF stack. Finally, the RobotCommand receives the TinyOS packet, processes this packet and drives the robot. We are able to demonstrate control of TinyOS robot using a Joystick. However, there is a time delay between the information read from a Joystick and the message arrive to the mica mote of our robot because of the MATLAB interface is not real time. This will improve as the Matlab to TinyOS interface improves.

5. Acknowledgements

I would like to thank Professor Kristofer Pister for giving me the chance to work with him and his group in his laboratory. I would also like to give a huge thank you to my mentor, Sarah Bergbreiter, for her help and her knowledge. Without her help, I would not be able to achieve my summer goal. I would also like to thank the entire stuff of the SUPERB program for organizing the program.

6. References

[1] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister. System Architecture Directions for Networked Sensors. In Architectural Support for Programming Languages and Operating Systems,2000

[2] Joseph Jones, Anita Flynn. Mobile Robots: inspiration to implementation Artificial Intelligence Laboratory, MIT, Wellesley, MA

[3] H. Jin Kim, Rene Vidal, David H. Shim, Omid Shankernia, Shankar Sastry. A Hierarchical Approach to Probabilistic Pursuit-Evasion Games with Unmanned Ground and Aerial Vehicles, Department of EECS, UC Berkeley.

[4] http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/tinyos/

[5] http://www-bsac.eecs.berkeley.edu/~sbergbre/CotsBots/specs.pdf

[6] http://today.cs.berkeley.edu/tos/design/mica/MICA_DESIGN_PAPER.doc

[7] http://www.mathworks.com/company/digest/june01/joystick.shtml