

Final Binder

Postlab 12

Revision 1

FLANCREST ENTERPRISES

CS 340 Group 22

May 3, 2002

Contents

1	About CS 340 and This Document	1
2	Mockup Requirements	2
2.1	GUI	2
2.2	Debugger	2
2.3	Robot Server	3
3	Debugger Requirements Specification	4
3.1	Introduction	4
3.1.1	Purpose of Software	4
3.1.2	Document Notation	4
3.2	Non-functional	5
3.2.1	Required Software	5
3.2.2	Required Hardware	5
3.2.3	Performance	5
3.3	Functional	5
3.3.1	Inputs	5
3.3.2	Outputs	7
3.3.3	Important Conditions	8
3.3.4	Modes	8
3.3.5	Mode Transitions	8
3.3.6	Events	8
3.3.7	Communications Protocol	11
3.4	Response to Undesired Events	11
3.4.1	Malformed Command over Network	11
3.4.2	Incorrect Command in the Enter Command text box	11
3.5	Fundamental Assumptions	11
3.6	Changes	11
3.7	Domain Terms	11
3.8	Sources	12
4	Robot Server Requirements Specification	13
4.1	Introduction	13
4.1.1	Purpose of Software	13

4.1.2	Document Notation	13
4.2	Non-functional	14
4.2.1	Required Software	14
4.2.2	Required Hardware	14
4.2.3	Performance	14
4.3	Functional	14
4.3.1	Inputs	14
4.3.2	Outputs	15
4.3.3	Modes Descriptions	17
4.3.4	Mode Transitions	17
4.3.5	Communications Protocol	17
4.4	Response to Undesired Events	17
4.5	Fundamental Assumptions	18
4.6	Changes	18
4.7	Sources	18
5	Protocol Specification Development Process	19
5.1	Process	19
5.2	Milestones	19
5.3	Schedule	20
6	Milestones	21
6.1	Milestones	21
7	Communication Protocol Specification	24
7.1	Introduction and Assumptions	24
7.1.1	Purpose	24
7.1.2	Capabilities	24
7.1.3	Assumptions	24
7.1.4	Document Notation	24
7.2	Items That Will Change	25
7.3	Functional	25
7.3.1	Introduction	25
7.3.2	Format	25
7.3.3	Commands	26
7.3.4	Status Queries	28
7.3.5	Responses	30
7.3.6	Persistence Messages	32
7.3.7	Errors & Warnings	32
7.4	Non-Functional	34
7.4.1	Performance	34
7.4.2	Transmission Medium	34
7.4.3	Security	34
7.4.4	Command Preemption	34

7.5	Undesired Events	35
7.6	Symbolic Constants	35
7.7	Non-Message Text Macros	35
7.8	Sources	35
7.9	Domain Terms	35
8	GUI Client Design	
	FANG	36
8.1	Introduction	36
8.1.1	Document Purpose	36
8.1.2	Statement of Project Objective	36
8.1.3	Document Notation	37
8.2	Architecture	37
8.2.1	Use	37
8.2.2	Objects	37
8.3	Static Structure	38
8.4	C++ Interfaces	38
8.4.1	fang/robot.h	38
8.4.2	fang/cmd.h	42
8.4.3	fang/gui.h	43
8.4.4	fang/script.h	45
8.4.5	fang/vjoystick.h	46
8.4.6	fang/cmdtimepair.h	47
8.5	Use Cases	48
8.5.1	Connect to the Robot	48
8.5.2	Disconnect from the Robot	48
8.5.3	Robot Movement	49
8.5.4	Stop the Robot	49
8.5.5	Turn the Robot while the Robot is stationary	50
8.5.6	Nudge the Robot in a desired direction	51
8.5.7	Change the Speed of the Robot	52
8.5.8	Pan the Camera	52
8.5.9	Tilt the Camera	53
8.6	Object Interactions	53
8.6.1	Sequence Diagrams	55
8.7	Script File Format	55
8.8	Possible Changes	55
8.9	References	56
9	Testing Report	58
9.1	Goals for Our Prototype	58
9.2	Milestones Set for This Week	59
9.3	Testing Activity	59
9.3.1	Testing with FURL	59

9.3.2	Testing with robot server	59
9.4	FANG's Source	60
10	System Performability	61
10.1	Performability Goals	61
10.2	Milestones	62
10.3	Integration Testing	62
10.4	Implementation Status	63
11	Implementation State	65
11.1	Implementation Functionality	65
11.2	Final Design vs Design Document	66
11.2.1	Not Present in Final Design	66
11.2.2	New in Final Design	66
11.2.3	Changed in Final Design	66
11.3	Changes Since Enhanced Prototype	67
11.4	Key Implementation Parameters	67
11.4.1	Lines of Code	67
11.4.2	Misc	67
11.5	ChangeLog	67
12	Example Management Report	71
12.1	Management Responsibilities	71
12.2	Previous Responsibilities	71
12.3	Meeting Reports	72
12.3.1	Meeting #1	72
12.3.2	Meeting #2	72
12.4	Activity Log	73
12.5	Schedule of Work Assignments	74
12.6	Unresolved Problems	74
13	Example Management Report	75
13.1	Management Responsibilities	75
13.2	Previous Postlab Responsibilities	75
13.3	Meeting Reports	76
13.3.1	Meeting #1	76
13.3.2	Meeting #2	76
13.4	Activity Log	77
13.5	Schedule of Work Assignments	79
13.6	Unresolved Problems	79
14	Example Management Report	80
14.1	Management Responsibilities	80
14.2	Previous Responsibilities	80
14.3	Meeting Reports	81

14.3.1 Meeting #1	81
14.4 Activity Log	82
14.5 Unresolved Problems	82
15 Example Management Report	83
15.1 Management Responsibilities	83
15.2 Previous Responsibilities	83
15.3 Meeting Reports	84
15.3.1 Meeting #1	84
15.4 Activity Log	85
15.5 Unresolved Problems	86
16 Management Report Template Source	87
16.1 Makefile	87
16.2 mngmnt_report.tex	88
16.3 preamble.tex	91
16.4 cmds_global.tex	92
16.5 cmds_mngmnt.tex	92
16.6 approval_pledge_toc.tex	94
16.7 meetings.tex	95
16.8 chris_activity.tex	95
16.9 emilio_activity.tex	96
16.10scott_activity.tex	96
16.11thomas_activity.tex	96
16.12chris_responsibilities.tex	96
16.13emilio_responsibilities.tex	96
16.14scott_responsibilities.tex	96
16.15thomas_responsibilities.tex	96
16.16problems.tex	96
17 Generic Document Template Source	97
17.1 generic_doc.tex	97
17.2 preamble.tex	98
17.3 cmds_global.tex	99
17.4 approval_pledge_toc.tex	100
18 Webpage Template Source	101
18.1 webpage.shtml	101
19 Screenshots of FANG	103
A Selected Website Pages	106
B FANG's Source	107

C FURL's Source	108
D Partner Group's Client Interface Specification	109

Approval

We, the members of Flancrest Enterprises, accept this document as representation of our work for the past week and agree that fair contributions to this project have been made by all whose signature appear below.

Name	Signature
Chris Frost	_____
Emilio Lahr-Vivaz	_____
Scott Rein	_____
Thomas Whanger	_____

Pledge

On our honor as students of University of Virginia, we pledge that we have neither given nor received aid on this assignment.

Name	Signature
Chris Frost	_____
Emilio Lahr-Vivaz	_____
Scott Rein	_____
Thomas Whanger	_____

Chapter 1

About CS 340 and This Document

Postlab 12
May 3, 2002

CS 340 is the second course in techniques for the development of software at The University of Virginia. The topics that were covered include: project management, scheduling, planning, configuration management, semi-formal and formal specification, object-oriented design, programming practices, application generation and component-based software reuse, and inspections.

CS 340 is organized around a semester-long project involving the development of the control software for a mobile robot. All laboratory activities are concerned with some aspect of the project. This binder includes a multitude of documents from many of the steps in the software development process.

Our group, Flancrest Enterprises, competed in the 2002 CS 340 Robot Games sponsored by Microsoft, in which teams compete in three events testing their system's capabilities: a timed obstacle course, dancing, and soccer. This competition allows for all the hard work put in to the project throughout the semester to pay off, and it certainly did. Flancrest Enterprises was the 2002 CS 340 Robot Games Champion.

Chapter 2

Mockup Requirements

Postlab 2
February 4, 2004

2.1 GUI

- in VB
- not functional
- status
 - sonar sensors
 - barrier info from sonar data
 - motor running
 - look at array info
- “simple and intuitive”
- keyboard and mouse input
- draw a path to follow

2.2 Debugger

- in VB
- act as a middle man between client and server (not necessary to have client connected)
- state vector
- current coordinate

- previous command history
- send commands
- abort
- emergency stop
- future: value of internal values (logging)
- use gdb for robot code debug

2.3 Robot Server

- give full access to movement and camera control
- report failures and warnings
- report sensor info
- accept basic cmds
evolve from “accelerate” to also “distance” and then more abstract commands
- stop if connection dies (after 1second of inactivity)
- stop if hit something (changeable at runtime)
- logging facility

Chapter 3

Debugger Requirements Specification

Postlab 3
May 3, 2002

3.1 Introduction

3.1.1 Purpose of Software

The program is used to debug the on-board robot software.

3.1.2 Document Notation

- The input variable foo is denoted: /foo/
- The output variable foo is denoted: //foo//
- A textmacro stands for a quantity derived from an input: !foo!
- A nonnumerical variables: \$foo\$
- A mode in which the software is in: *foo*
- A term specific to the problem domain: *foo*
- Do something when /foo/ becomes equal to !bar!: @T(/foo/=!bar!)
- Do above, but only if !baz! is equal to /bert/: @T(/foo/=!bar!) WHEN (!baz!=/bert/)
- Do something when /foo/ becomes not equal to !bar!: @F(/foo/=!bar!)
- Do above, but only if !baz! is equal to /bert/: @F(/foo/=!bar!) WHEN (!baz!=/bert/)

3.2 Non-functional

3.2.1 Required Software

The program should be able to run on windows and interface with the on-board robot software via TCP.

3.2.2 Required Hardware

The software must conform to our communication protocol specification, responding to all messages as indicated.

3.2.3 Performance

The software must respond in a reasonable amount of time to control the robot. The software must respond within the specified *response time*.

3.3 Functional

3.3.1 Inputs

Input Data Item: Execute Command *Button*
/EXECOM/

Description: Whether the *button* is On or Off
Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Halt Program *Button*
/HALTPROG/

Description: Whether the *button* is On or Off
Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Resume *Button*
/RESUME/

Description: Whether the *button* is On or Off
Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Single Step *Button*
/SNGLSTEP/

Description: Whether the *button* is On or Off
Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Abort *Button*
/ABORT/

Description: Whether the *button* is On or Off

Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Clear *Button*

/CLEAR/

Description: Whether the *button* is On or Off

Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Save *Button*

/SAVE/

Description: Whether the *button* is On or Off

Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Command Text

/COMTEXT/

Description: A string of text that can be interpreted as a command to the robot server

Characteristics of Values: Any valid network command

Input Data Item: Command Center *Check Box*

/CMDCNTCHK/

Description: Whether the *Check Box* is Checked or Unchecked

Characteristics of Values: \$Checked\$ \$Unchecked\$

Input Data Item: Command Log *Check Box*

/CMDLOGCHK/

Description: Whether the *Check Box* is Checked or Unchecked

Characteristics of Values: \$Checked\$ \$Unchecked\$

Input Data Item: Variable State Array *Check Box*

/VARCHK/

Description: Whether the *Check Box* is Checked or Unchecked

Characteristics of Values: \$Checked\$ \$Unchecked\$

Input Data Item: Command Center *X Box*

/CMDCNTX/

Description: Whether the *X Box* is On or Off

Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Command Log *X Box*

/CMDLOGX/

Description: Whether the *X Box* is On or Off

Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Variable State Array *X Box*

/VARX/

Description: Whether the *X Box* is On or Off

Characteristics of Values: \$On\$ \$Off\$

Input Data Item: Connect *Button*

/CONNECT/

Description: Whether the *button* is On or Off

Characteristics: \$On\$ \$Off\$

Input Data Item: Host Text Box

/HOST/

Description: A string of characters that represents a host to be connected to.

Characteristics: Any valid hostname or IP address

Input Data Item: Port Text Box

/PORT/

Description: A string of characters that represents a port on the host

Characteristics: Any valid port on the host

3.3.2 Outputs

Output Data Item: Variable State Array

//VARARRAY//

Description: The 30 outputs of the sensor and variable information coming from the robot server

Characteristics of Values:

Range: 8 bit

Output Data Item: Command Log

//CMDLOG//

Description: The command log executed by the robot server

Characteristics of Values: Any sequence of valid robot command

Output Data Item: Command

//CMD//

Description: A command to be executed by the robot server

Characteristics of Values: Any valid robot command

Output Data Item: Coordinates

//COORD//

Description: The current coordinates of the robot

Characteristics of Values:

Unit: Inches?

3.3.3 Important Conditions

/LOG/ := T iff (/CMDLOGCHK/ = \$Checked\$)
 /VAR/ := T iff (/VARCHK/ = \$Checked\$)

3.3.4 Modes

Mode	Cond
Initial	/VARCHK/ = \$Unchecked\$ AND /CMDLOGCHK/ = \$Unchecked\$
Log	/VARCHK/ = \$Unchecked\$ AND /CMDLOGCHK/ = \$Checked\$
Update	/VARCHK/ = \$Checked\$ AND /CMDLOGCHK/ = \$Unchecked\$
Full	/VARCHK/ = \$Checked\$ AND /CMDLOGCHK/ = \$Checked\$

3.3.5 Mode Transitions

Initial to *Log* — @T(/LOG/)
 Initial to *Update* — @T(/VAR/)
 Log to *Full* — @T(/VAR/)
 Update to *Full* — @T(/LOG/)
 Full to *Log* — @F(/VAR/)
 Full to *Update* — @F(/LOG/)
 Log to *Initial* — @F(/LOG/)
 Update to *Initial* — @F(/VAR/)

3.3.6 Events

Periodic Function Name: Robot Variables

Modes: *Full* or *Update*

Output data item: //VARARRAY//

Conditions: Always

Updates all the variables in //VARARRAY//

Periodic Function Name: Robot Coordinates

Modes: All

Output data item: //COORD//

Conditions: Always

Updates the coordinates in //COORD//

Periodic Function Name: Heartbeat

Modes: All

Output data item: none

Conditions: Always

Sends an heartbeat to the server according to the communication protocol

Demand Function Name: Command Log

Modes: *Log* or *Full*

Output data item: //CMD//

Conditions: @T(Receipt of command execution)

Records the command in the log

Demand Function Name: Log window Open

Modes: *Update* or *Initial*

Output data item: none

Conditions: @T(/CMDLOGCHK/ = \$On\$)

Opens the command log window

Demand Function Name: Log window Close

Modes: *Full* or *Log*

Output data item: none

Conditions: @F(/CMDLOGCHK/ = \$Off\$) OR @F(/CMDLOGX/ = \$On\$)

Closes the command log window, sets /CMDLOGX/ = \$Off\$, sets /CMDLOGCHK/ = \$Off\$

Demand Function Name: Variable window Open

Modes: *Log* or *Initial*

Output data item: none

Conditions: @T(/VARCHK/ = \$On\$)

Opens the variable state window

Demand Function Name: Variable window Close

Modes: *Full* or *Update*

Output data item: none

Conditions: @F(/VARCHK/ = \$Off\$) OR @F(/VARX/ = \$On\$)

Closes the variable state window, sets /VARX/ = \$Off\$, sets /VARCHK/ = \$Off\$

Demand Function Name: Command center window Open

Modes: All Output data item: none

Conditions: @T(/CMDCNTCHK/ = \$On\$)

Opens the command center window

Demand Function Name: Command center window Close

Modes: All Output data item: none

Conditions: @F(/CMDCNTCHK/ = \$Off\$) OR @F(/CMDCNTX/ = \$On\$)

Closes the command center window, sets /CMDCNTX/ = \$Off\$, sets /CMDC-

NTCHK/ = \$Off\$

Demand Function Name: Abort program

Modes: All Output data item: none

Conditions: @T(/ABORT/ = \$On\$)

Stops all actions of the program, sets /ABORT/ = \$Off\$

Demand Function Name: Halt program

Modes: All Output data item: none

Conditions: @T(/HALTPROG/ = \$On\$) AND /CMDCNTCHK/ = \$On\$

Stops commands sent to the robot, sets /HALTPROG/ = \$Off\$

Demand Function Name: Execute command

Modes: All Output data item: none

Conditions: @T(/EXECOM/ = \$On\$) AND /CMDCNTCHK/ = \$On\$

Executes command in /COMTEXT/, sets /EXECOM/ = \$Off\$

Demand Function Name: Single step command

Modes: All Output data item: none

Conditions: @T(/SNGLSTEP/ = \$On\$) AND /CMDCNTCHK/ = \$On\$

Executes next command then halts, sets /SNGLSTEP/ = \$Off\$

Demand Function Name: Resume program

Modes: All Output data item: none

Conditions: @T(/RESUME/ = \$On\$) AND /CMDCNTCHK/ = \$On\$

Resumes sending of commands to the robot, sets /RESUME/ = \$Off\$

Demand Function Name: Save log

Modes: *Log* or *Full* Output data item: //CMDLOG//

Conditions: @T(/SAVE/ = \$On\$)

Saves the log data to an external file, sets /SAVE/ = \$Off\$

Demand Function Name: Clear log

Modes: *Log* or *Full* Output data item: //CMDLOG//

Conditions: @T(/CLEAR/ = \$On\$)

Erases the data in the log, sets /CLEAR/ = \$Off\$

Demand Function Name: Connect to Host

Modes: All

Output data item: none

Conditions: @T(/CONNECT/ = \$On\$)

Connects to the host specified by /HOST/ on the port specified by /PORT/, sets /CONNECT/ = \$Off\$

3.3.7 Communications Protocol

See sources.

3.4 Response to Undesired Events

3.4.1 Malformed Command over Network

If a command received over the network is incorrect, the program will ignore it because the TCP/IP should handle this.

3.4.2 Incorrect Command in the Enter Command text box

If an incorrect command is typed into the text box in the Command Center window, and the user hits the Execute *button*, the program will bring up a message box. This message box will simply tell the user the command was incorrect.

3.5 Fundamental Assumptions

We assume that the program will stop running if the main window is ever closed. We further assume that input *buttons* can only be activated if their associated window is open. We assume that the program will communicate via TCP, and thus TCP will handle the transmission errors.

3.6 Changes

3.7 Domain Terms

Communications Protocol — see Communications Protocol Specification.

Buttons — Visual input devices that can be clicked by the user to trigger a function. Once clicked, they reset so that they can be clicked again. *Buttons* are \$Off\$ until the *button* is depressed and released, at which time they become \$On\$.

Check Boxes — Visual input devices that can be clicked by the user to trigger a function. Once clicked, they stay activated until clicked again.

X Boxes — Visual input devices that can be clicked by the user to trigger a function. Once clicked, they reset so that they can be clicked again. *X Boxes* are \$Off\$ until the *X Boxes* is depressed and released, at which time they become \$On\$. *X Boxes* are essentially *buttons*, but with the specific function of closing their associated window.

Response Time — A constant amount of time within which the software must execute a function after it is called by the user. The actual amount of time will not be specified here, but it should be no greater (and probably much less) than 1 second.

3.8 Sources

Emilio Lahr-Vivaz

Thomas Whanger

“Communications Protocol Specification” by Flancrest Enterprises and Decepticon Industries

Chapter 4

Robot Server Requirements Specification

Postlab 3
May 3, 2002

4.1 Introduction

4.1.1 Purpose of Software

The purpose of this software, the *robot server*, is to allow a remote *client* to control and query a robot and to abstract the lower details for this client, such as the actual api calls. It should listen for commands via a TCP/IP connection, perform the indicated actions, and send specific information to the client as this document details.

4.1.2 Document Notation

- The input variable foo is denoted: /foo/
- The output variable foo is denoted: //foo//
- A textmacro stands for a quantity derived from an input: !foo!
- A nonnumerical variables: \$foo\$
- A mode in which the software is in: *foo*
- A term specific to the problem domain: *foo*
- Do something when /foo/ becomes equal to !bar!: @T(/foo/=!bar!)
- Do above, but only if !baz! is equal to /bert/: @T(/foo/=!bar!) WHEN (!baz!=/bert/)

- Do something when `/foo/` becomes not equal to `!bar!`: `@F(/foo/!=!bar!)`
- Do above, but only if `!baz!` is equal to `/bert/`: `@F(/foo/!=!bar!) WHEN (!baz!=/bert/)`

4.2 Non-functional

4.2.1 Required Software

- Nomad *API*
- GCC compiler v2.95.2
- Flancrest and Decepticon GUI/debugger *client* software
- *TCP/IP* network connection
- RedHat Linux v6.1

4.2.2 Required Hardware

- Compatible Nomad Scout *robot*
- Network interface on *robot*

4.2.3 Performance

- Process at least 10 commands per second (from client to the robot)

4.3 Functional

4.3.1 Inputs

`/COMM-MSG/`

Description: Data sent from client via communications protocol. See *Communications Protocol Specification* by Flancrest Enterprises and Decepticon Industries. Commands should be queued if they cannot be acted upon immediately.

`/STATE-SONAR-0/ ... /STATE-SONAR-15/`

Description: Sonar data, giving distance to nearest object

Range: **TODO:** Unknown

Units: inches

`/STATE-CONF-X/`

Description: x position of the *robot*

Range:

Units: 1/10 inches

/STATE-CONF-Y/

Description: y position of the *robot*

Range:

Units: 1/10 inches

/STATE-CONF-STEER/

Description: steering angle

Range: [0, 3600]

Units: 1/10 degrees

/STATE-CONF-TURRET/

Description: turret angle

Range: [0, 3600]

Units: 1/10 degrees

/STATE-VEL-TRANS/

Description: translational velocity

Range: [0, 200]¹

Units: 1/10 inch/sec

/STATE-VEL-TURRET/

Description: turret velocity

Range: [0, 450]²

Units: 1/10 degree/sec

//SMASK-SONAR-0// ... //SMASK-SONAR-15//, //SMASK-CONF-X//, //SMASK-CONF-Y//, //SMASK-CONF-STEER//, //SMASK-CONF-TURRET//, //SMASK-VEL-TRANS//, //SMASK-VEL-STEER//, and //SMASK-VEL-TURRET//

Description: masks that enable/disable updating of information

Range: [0, 1]

Units: !Update! := 1, !¬Update! := 0

4.3.2 Outputs

//CLIENT//

Description: Data sent to client via communications protocol. See *Communications Protocol Specification* by Flancrest Enterprises and Decepticon Industries.

¹Is this correct? Robot documentation not clear.

²Is this correct? Robot documentation not clear.

//STATE-SONAR-0// ... //STATE-SONAR-15//

Description: Sonar data, giving distance to nearest object

Range: **TODO:** Unknown

Units: inches

//STATE-CONF-X//

Description: x position of the robot

Range:

Units: 1/10 inches

//STATE-CONF-Y//

Description: y position of the robot

Range:

Units: 1/10 inches

//STATE-CONF-STEER//

Description: steering angle

Range: [0, 3600]

Units: 1/10 degrees

//STATE-CONF-TURRET//

Description: turret angle

Range: [0, 3600]

Units: 1/10 degrees

//STATE-VEL-TRANS//

Description: translational velocity

Range: [0, 200]³

Units: 1/10 inch/sec

//STATE-VEL-STEER//

Description: steering velocity

Range: [0, 450]⁴

Units: 1/10 degree/sec

//STATE-VEL-TURRET//

Description: turret velocity

Range: [0, 450]⁵

Units: 1/10 degree/sec

³Is this correct? Robot documentation not clear.

⁴Is this correct? Robot documentation not clear.

⁵Is this correct? Robot documentation not clear.

//SMASK-SONAR-0// ... //SMASK-SONAR-15//, //SMASK-CONF-X//, //SMASK-CONF-Y//, //SMASK-CONF-STEER//, //SMASK-CONF-TURRET//, //SMASK-VEL-TRANS//, //SMASK-VEL-STEER//, and //SMASK-VEL-TURRET//

Description: masks that enable/disable updating of information

Range: [0, 1]

Units: !Update! := 1, !¬Update! := 0

4.3.3 Modes Descriptions

Mode	Description	Action
NORMAL	Normal operation of <i>robot</i> <i>server</i>	Issue commands to <i>robot</i> as they are received and queued, and provide feedback to <i>client</i>
NO-CONN	No connection established	Listen for <i>client</i>
EMER-STOP	Emergency stop of <i>robot</i>	Stop <i>robot</i> motors and clear !queue!

4.3.4 Mode Transitions

Transition	Event
NORMAL to *EMER-STOP*	@T(!TimeSinceLastHeartbeat! ≥ !HeartbeatTimeout!) WHEN (!NetworkConnected!)
EMER-STOP to *NORMAL*	@T(!Reset!)
NORMAL to *NO-CONN*	@F(!NetworkConnected!)
NO-CONN to *NORMAL*	@T(!NetworkConnected!)

4.3.5 Communications Protocol

This software must conform to the requirements specified by *Communications Protocol Specification* by Flancrest Enterprises and Decepticon Industries.

4.4 Response to Undesired Events

!ConnTimeout!

Description: How long after a connection has been lost until the server should disconnect and rebegin listening to incoming connections.

Value: !ConnTimeout! := 15 seconds > !HeartbeatTimeout!

!HeartbeatTimeout!

Description: How long after not hearing a heartbeat the robot should switch into *EMER-STOP*.

- @T(!TimeSinceLastHeartbeat! ≥ !HeartbeatTimeout!): Switch into *EMER-STOP*.

- @T(!TimeSinceConnActivity! \geq !ConnTimeout!): Switch into *NO-CONN*.
- Bumper gets hit:
 - @T(!BMPR-HIT!) WHEN (!BMPR-STOPS! = T): Stop immediately, clear the command queue and wait for the client to signal a restart.
 - @T(!BMPR-HIT!) WHEN (!BMPR-STOPS! = F): Do not stop, continue.
- Malformed messages (and out of range): Follow the directions given in *Communications Protocol Specification*.
- Sensors have problems: Send data to !CLIENT! using the given protocol.

4.5 Fundamental Assumptions

- Using Nomad *API* for *robot* control
- Running on a laptop connected via serial port to *robot*
- Compatible client software to send commands for *robot* to execute
- Reliable medium (*TCP/IP* network) to communicate with *client*

4.6 Changes

- More commands

4.7 Sources

- Client \Rightarrow Server communications protocol: *Communications Protocol Specification* by Flancrest Enterprises and Decepticon Industries, Chris Frost, or Keen Browne.
- Robot hardware specifics: Scott Rein, Chris Frost, or Nomadic Technologies, Inc.'s documentation.

Chapter 5

Protocol Specification Development Process

Postlab 3
May 3, 2002

5.1 Process

1. Come up with simple high level requirements for protocol
2. Individuals meet and discuss details of requirements
 - message formats
 - message size
 - constraints on communication
 - message capabilities
3. Individuals write specification draft
 - Using natural language and A7E
4. Draft returns to individual groups and is reviewed
5. Return to step 3 and repeat until all parties are satisfied
6. Turn in final protocol specification

5.2 Milestones

- High level requirements completed
- Draft 1 of specification completed

- Draft 1 of specification reviewed
- \vdots
- Draft n of specification completed
- Draft n of specification reviewed
- Final specification completed

5.3 Schedule

Date Due:	2/3	2/11	2/15	2/16	3/11
Step:	1	2	3	4	6

Chapter 6

Milestones

Prelab 5
May 3, 2002

6.1 Milestones

- Sign off the document describing the completed implementation of the support tool *Due: 03-04-22, Prelab 6*
 - Design
 - * Understand requirements specification
 - * Choose classes, algorithms, and file formats
 - * Document decisions
 - Implement
 - * Clear the mind
 - * Divide up work among members
 - Document implementation
- Final version of communication protocol specification signed off *Due: 03-04-02, Postlab 6*
 - Both groups discuss fundamental requirements
 - Preliminary draft
 - * Organize skeleton document
 - * Write sections in combination natural language and a7e
 - Second draft and revisions
 - * Give preliminary draft to partner group for critique
 - * Analyze internally
 - * Correct found problems

- Final document and revisions
 - * Repat previous until all parties satisfied
- Sign off a comprehensive design document *Due: 03-18-02, Postlab 7*
 - Brainstorm design
 - UML class diagram
 - * Research UML
 - C++ class interfaces
 - State and sequence diagrams
 - Inspect the document
 - * Create checklist
 - * Inspect
- Show evolutionary prototype to customer *Due: 03-18-02, Prelab 8*
 - Define and document the goals for implementing a demo of the robot system
 - * Decide what would be most helpful to show the customer
 - * Decide what is realistic for this stage of development
 - Build the prototype
 - Test the prototype
 - * Create tests to check software
- Perform basic performability analysis of robot-control system *Due: 03-25-02, Prelab 9*
 - Define schedule and milestones for the enhanced prototype
 - * Reanalyze future work
 - * Redfine milestones
 - Develop the enhanced prototype
 - * Address needs of performability analysis
 - Conduct a variety of functional tests
 - * Design tests
 - * Analyze results
- Complete preliminary implementation *Due: 04-01-02, Prelab 10*
 - Complete the implementation of the robot-control system according to the original specification

- * Complete in iterative steps
 - * Split up and implement
- Identify any new risks
- Identify areas of design that need to be modified
 - * Ask questions of ourselves
 - * Ask random people walking by
- Develop the design
- Inspect partner group's code *Due: 04-8-02, Prelab 11*
 - Develop an inspection checklist
 - * Develop items separately and bring lists together
 - Conduct three phase inspection
 - Conduct necessary rework before the next stage in inspection
 - * Address found issues
 - Document inspection
- Final Specification *Due: 04-22-02, Postlab 12*
 - Finalize specification
 - * Change document to correspond to all changes in specification
- Compile all documents *Due: 04-22-02, Postlab 12*
 - Integrate all documents
 - * Convert L^AT_EX articles to a single book
 - Take over Stacks to print out all documents

Chapter 7

Communication Protocol Specification

Postlab 6
March 18, 2002

7.1 Introduction and Assumptions

7.1.1 Purpose

This communication protocol should allow the *robot server* software and the *user interface client* of Decepticon Industries and Flancrest Enterprises to communicate information. This information is robot commands, status queries from the client, responses from the robot, errors, and warnings.

7.1.2 Capabilities

This communication protocol specification details commands to move the *robot* and its *camera*, get status information and communicate warnings. This specification requires commands to have a command name and a variable number of arguments.

7.1.3 Assumptions

- The *robot* has similar functionality to the Nomad Scout.
- The *user* is interacting with the robot through a *user interface* not on the robot.

7.1.4 Document Notation

- The input variable foo is denoted: /foo/
- The output variable foo is denoted: //foo//

- A textmacro stands for a quantity derived from an input: `!foo!`
- A nonnumerical variables: `foo`
- A mode in which the software is in: `*foo*`
- A term specific to the problem domain: `foo`
- Do something when `/foo/` becomes equal to `!bar!`: `@T(/foo/=!bar!)`
- Do above, but only if `!baz!` is equal to `/bert/`: `@T(/foo/=!bar!) WHEN (!baz!=/bert/)`
- Do something when `/foo/` becomes not equal to `!bar!`: `@F(/foo/=!bar!)`
- Do above, but only if `!baz!` is equal to `/bert/`: `@F(/foo/=!bar!) WHEN (!baz!=/bert/)`

7.2 Items That Will Change

- Additional commands, both lower and higher level

7.3 Functional

7.3.1 Introduction

This set of messages includes the functions available on the Nomadic Technologies, Inc. *robot* and additional messages to describe status reported by the robot. Messages fall into two categories. Commands tell the robot to move, and status queries tell the robot to reply with certain status information. Replies from the robot to the user interface also fall into two categories: responses and errors and warnings. Responses are replies to status queries. Errors and warnings are messages from the robot that advise the user interface of some problem. The message format must describe multiple types of messages with an arbitrary number of parameters. In the following section each of the messages necessary for communication between the user interface and the robot are described. This list may change and the software must be capable of handling new messages.

7.3.2 Format

Messages between the robot server and the user interface client are sent as text. The text message is delimited by parentheses. The command and arguments in the text message are separated by spaces. Arguments must be passed in the order in which they are listed in the tables in this document.

Generic Format: `(command argument1 argument2 ... argument n)`

Example: `(vm 1 1)`

7.3.3 Commands

Robot Movement

Command:

da

Description:

Defines the robot's *steering angle*. Angles increase in the counterclockwise direction.

Argument	Units	Range
theta	1/10ths of a degree	[0, 3600]

Command:

pr

Description:

moves the motors of the robot by a distance, using the previously set speed.

Argument	Units	Range
right wheel step	1/10 inches	[-32000, 32000]
left wheel step	1/10 inches	[-32000, 32000]

Command:

rotateRobot

Description:

Rotates the robot by the given degrees. Angles increase in the counterclockwise direction.

Argument	Units	Range
theta	1/10ths of a degree	[-3600, 3600]

Command:

sp

Description:

sets the left and right wheel speeds of the robot.

Argument	Units	Range
Right Wheel Speed	1/10th inch/sec	[0, 400]
Left Wheel Speed	1/10th inch/sec	[0, 400]

Command:

st

Description:

Stops the motion of the *robot base* and *camera*

Argument	Units	Range
robot	1: yes, 0: no	0 or 1
camera	1: yes, 0: no	0 or 1

Command:

vm

Description:

moves the robot at a specified velocity.

Argument	Units	Range
right wheel velocity	1/10th inch/sec	[-400, 400]
left wheel velocity	1/10th inch/sec	[-400, 400]

Command:

setStopOnBump

Description:

Tell robot whether to stop or not when the bump switch is hit. A parameter value of 0 means stop, and a parameter value of 1 means start

Argument	Units	Range
flag	none	[0, 1]

Command:

reset

Description:

If !stoponbump! is true and bumper has been hit, reenable robot control.

Argument	Units	Range
none		

Camera Movement

Command:

moveCamera

Description:

Move the camera at a *pan* and *tilt* value speed in \$angleunits\$. See section 7.3.3 for the command to stop the camera.

Argument	Units	Range
pan speed	1/10ths of degree/sec	[−450, 450]
tilt speed	1/10ths of degree/sec	[−450, 450]

Command:

bumpCamera

Description:

Move the camera a given distance at a given speed.

Argument	Units	Range
Pan Distance	1/10th of degrees	[−32000, 32000]
Tilt Distance	1/10th of degrees	[−32000, 32000]
Pan Rotational speed	1/10ths of degree/sec	[0, 450]
Tilt Rotational speed	1/10ths of degree/sec	[0, 450] TODO: Why not negative?

7.3.4 Status Queries

Command:

init

Description:

Initiates a connection between the user interface and the robot server software. The robot should ignore all commands until this command is received.

Argument	Units	Range
Right Wheel Speed	1/10th inch/sec	[0, 400]
Left Wheel Speed	1/10th inch/sec	[0, 400]
Pan Camera Speed	1/0ths of degree/sec	[0, 400] TODO: Why not negative?
Tilt Camera Speed	1/0ths of degree/sec	[0, 400]
Stop on Bump	1: yes, 0: no	0 or 1
heartBeatRate	10's of milliseconds	[\$minHeartBeat\$, \$maxHeartBeat\$]

Command:

disconnect

Description:

Discontinues a connection between the user interface and the robot server software. The robot should only accept this disconnect command after receiving the init command.

Argument	Units	Range
none		

Command:

getBp

Description:

Get the bumper data. A period of zero requests only one bumper data update and stops a currently-running periodic bumper data.

Argument	Units	Range
Period	10's of milliseconds	$[0, 2^{31} - 1]$

Command:

getRc

Description:

Get the *configuration data* of the robot. A period of zero requests only one configuration data update and stops a currently-running periodic configuration data.

Argument	Units	Range
Period	10's of milliseconds	$[0, 2^{31} - 1]$

Command:

getSp

Description:

Get the *left wheel*, *right wheel*, and *camera* velocities set by the sp or init commands. A period of zero requests only one velocities update and stops a currently-running periodic velocities.

Argument	Units	Range
Period	10's of milliseconds	$[0, 2^{31} - 1]$

Command:

getSn

Description:

Get the sonar data from the robot. A period of zero requests only one sonar data update and stops a currently-running periodic sonar data.

Argument	Units	Range
Period	10's of milliseconds	$[0, 2^{31} - 1]$

Command:

getState

Description:

Get all of the current state of the robot. A period of zero requests only one state data update and stops a currently-running periodic state data.

Argument	Units	Range
Period	10's of milliseconds	$[0, 2^{31} - 1]$

7.3.5 Responses

Command:

bpData

Description:

Information about which bumpers have been depressed. A value of 0 means the bumper is not depressed; a value of 1 means the bumper is depressed.

Argument	Units	Range
State of bumper zero	none	0 or 1
State of bumper one	none	0 or 1
State of bumper two	none	0 or 1
State of bumper three	none	0 or 1
State of bumper four	none	0 or 1
State of bumper five	none	0 or 1

Command:

snData

Description:

Returns the distance from all sonar sensors to detected object.

Argument	Units	Range
Sonar zero	inches	[6, 420]
Sonar one	inches	[6, 420]
Sonar two	inches	[6, 420]
Sonar three	inches	[6, 420]
Sonar four	inches	[6, 420]
Sonar five	inches	[6, 420]
Sonar six	inches	[6, 420]
Sonar seven	inches	[6, 420]
Sonar eight	inches	[6, 420]
Sonar nine	inches	[6, 420]
Sonar ten	inches	[6, 420]
Sonar eleven	inches	[6, 420]
Sonar twelve	inches	[6, 420]
Sonar thirteen	inches	[6, 420]
Sonar fourteen	inches	[6, 420]
Sonar fifteen	inches	[6, 420]

Command:

spData

Description:

Returns the speed of the robot.

Argument	Units	Range
Right Wheel	1/10th inches/sec	[−400, 400]
Left Wheel	1/10th inches/sec	[−400, 400]
Camera Pan	1/10th inches/sec	[0, 400] TODO: Why not negative?
Camera Tilt	1/10th inches/sec	[0, 400] TODO: Why not negative?

Command:

ack

Description:

Acknowledgement that specified a command has been completed.

Argument	Units	Range
Command Name	none	any command opcode

7.3.6 Persistence Messages

TODO: Should one side only send a heartBeat in response to the other?

Command:

heartBeat

Description:

This message is sent at \$heartBeatRate\$ intervals by both the robot and the user interface. Upon receiving a heartBeat from the other side, the receiving side must send a heartBeat back within the time given by \$heartBeatRate\$.

Argument	Units	Range
<i>none</i>		

Command:

setHeartBeat

Description:

Sets the \$heartBeat\$ period. A period of zero stops heartBeat activity.

Argument	Units	Range
Period	10's of milliseconds	[\$minHeartBeat\$, \$maxHeartBeat\$]

7.3.7 Errors & Warnings

Command:

bump

Description:

indicates a bumper is depressed.

Argument	Units	Range
Bumper Number	none	[0, 5]

Command:

speedMax

Description:

the left or right motors are at maximum speed

Argument	Units	Range
none		

Command:

panMax

Description:

camera is at maximum pan

Argument	Units	Range
none		

Command:

tiltMax

Description:

camera is at maximum tilt

Argument	Units	Range
none		

Command:

unknownCmd

Description:

Command is unknown

Argument	Units	Range
none		

Command:

invalidArgs

Description:

Invalid arguments passed with a command

Argument	Units	Range
none		

7.4 Non-Functional

7.4.1 Performance

Bandwidth

The protocol is designed assuming an available bandwidth of at least \$MinBandwidth\$

Command Rate

The commands in this protocol must allow at least \$minAvgCmdRate\$ rate of commands sent between the user interface client and robot server.

7.4.2 Transmission Medium

The protocol operates over a tcp/ip connection.

7.4.3 Security

The environment in which this protocol will be used is assumed to be closed off from the rest of the world, and so no special attention is required for security concerns. Thus there should be no user or system authentication or protection against rogue hosts on the network.

7.4.4 Command Preemption

The robot server must ignore conflicting commands sent which affect hardware currently being used. For example, if the robot is currently moving forward, a command to move backwards would be ignored. However, a command to move the camera or a request for status information would not. These command classes are independent:

- Robot movement
- Camera movement
- Status queries

7.5 Undesired Events

- Connection loss: The robot must stop its motors one half of a second after the *heartbeat* or connection is lost, clearing the command queue.
- Malformed/nonexistent command: Inform *client* and continue as if the command was not received.

7.6 Symbolic Constants

<code>\$minBandwidth\$</code>	25 kbps
<code>\$minHeartBeatRate\$</code>	0 heart beats / sec
<code>\$maxHeartBeatRate\$</code>	10 heart beats / sec
<code>\$minAvgCmdRate\$</code>	10 commands / sec

7.7 Non-Message Text Macros

<code>!heartBeatRate!</code>	Set by client at connection initiation, <code>!heartBeatRate!</code> \in [<code>\$minHeartBeatRate\$</code> , <code>\$maxHeartBeatRate\$</code>]
------------------------------	--

7.8 Sources

- Robot hardware questions: Nomadic Technologies' Nomad documentation
- Protocol questions: Keen Browne or Chris Frost

7.9 Domain Terms

Term	Definition
robot server	The software which runs on the laptop directly connected to the Nomad robots allowing the <i>user interface client</i> to connect and command the robot.
user interface client	Connects to the <i>robot server</i> , allowing the user to control the robot.
camera	A camera on the robot sending signals back to a tv via a wireless connection. Moveable by the robot.
robot	A Nomad Scout robot.
bumper	one of the five bumper sensors on the robot. When the robot collides with an object the bumper is triggered

Chapter 8

GUI Client Design *FANG*

Postlab 7
March 26, 2002

8.1 Introduction

8.1.1 Document Purpose

The purpose of this to describe the design of FANG. This document covers system architecture and high-level design.

8.1.2 Statement of Project Objective

FANG is the Flancrest Client GUI, providing a user interface to the the robot control system being developed by Flancrest Enterprises and Decepticon Industries according to FANG's requirements [1] and specification [2].

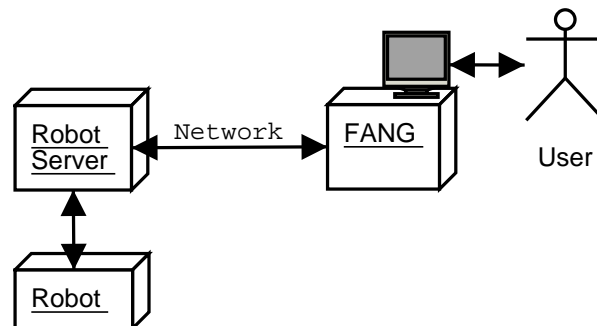


Figure 8.1: The robot system's major components' interactions

8.1.3 Document Notation

- The input variable foo is denoted: /foo/
- The output variable foo is denoted: //foo//
- A textmacro stands for a quantity derived from an input: !foo!
- A nonnumerical variables: \$foo\$
- A mode in which the software is in: *foo*
- A term specific to the problem domain: *foo*
- Do something when /foo/ becomes equal to !bar!: @T(/foo/=!bar!)
- Do above, but only if !baz! is equal to /bert/: @T(/foo/=!bar!) WHEN (!baz!=/bert/)
- Do something when /foo/ becomes not equal to !bar!: @F(/foo/=!bar!)
- Do above, but only if !baz! is equal to /bert/: @F(/foo/=!bar!) WHEN (!baz!=/bert/)

8.2 Architecture

8.2.1 Use

FANG is a graphical user interface, providing a means for a person to control a robot running Decepticon Industries's robot server¹ via our communications protocol [5]. Looking at FANG from an event perspective, there are two primary sources of events:

- User manipulation of the interface
- Incoming messages from the robot server

Use cases in section 8.5 provide an in-depth event perspective.

8.2.2 Objects

The architecture of FANG is broken up into two primary components: the GUI and Robot classes. The GUI class handles interactions with all user interface controls and thus the user. The Robot class is used by the GUI to tell the robot to perform actions and to request information. The Robot class's interface is neutral to any particular communication medium. The GUI and Robot classes use instances of the Cmd class to pass commands between each other. Sections 8.3 and 8.4 give a more in-depth analysis of the components making up FANG, section 8.6 details interactions among components.

¹Robot server: requirements [3] and specification [4].

8.3 Static Structure

Static structure in UML is shown in figure 8.2, page 39.

8.4 C++ Interfaces

8.4.1 fang/robot.h

```
// $Id: robot.h,v 1.7 2002/03/27 02:28:28 ccf7f Exp $
// Flancrest Enterprises, Group 22
// About this file:
// This class hides design decisions involving how information is sent to
// and received from the robot server (which hides how the actual hardware
// interfaces). In fact, this class completely abstracts the functional
// aspects of there existing a robot server.

#ifndef ROBOT_H
#define ROBOT_H

#include <string>
#include <afxmt.h>
#include "cmd.h"
#include "ws-util.h"
using namespace std;

class CFangDlg;

typedef DWORD timeSpan;    // milliseconds
typedef DWORD currentTime; // milliseconds

const bool debugPrintsCmds = true;

class Robot {
// Associations
private:
    CFangDlg *gui;

// Attributes
private:
    string hostname;
    WSADATA wsaData;
    SOCKET sd;
    const int port;
```

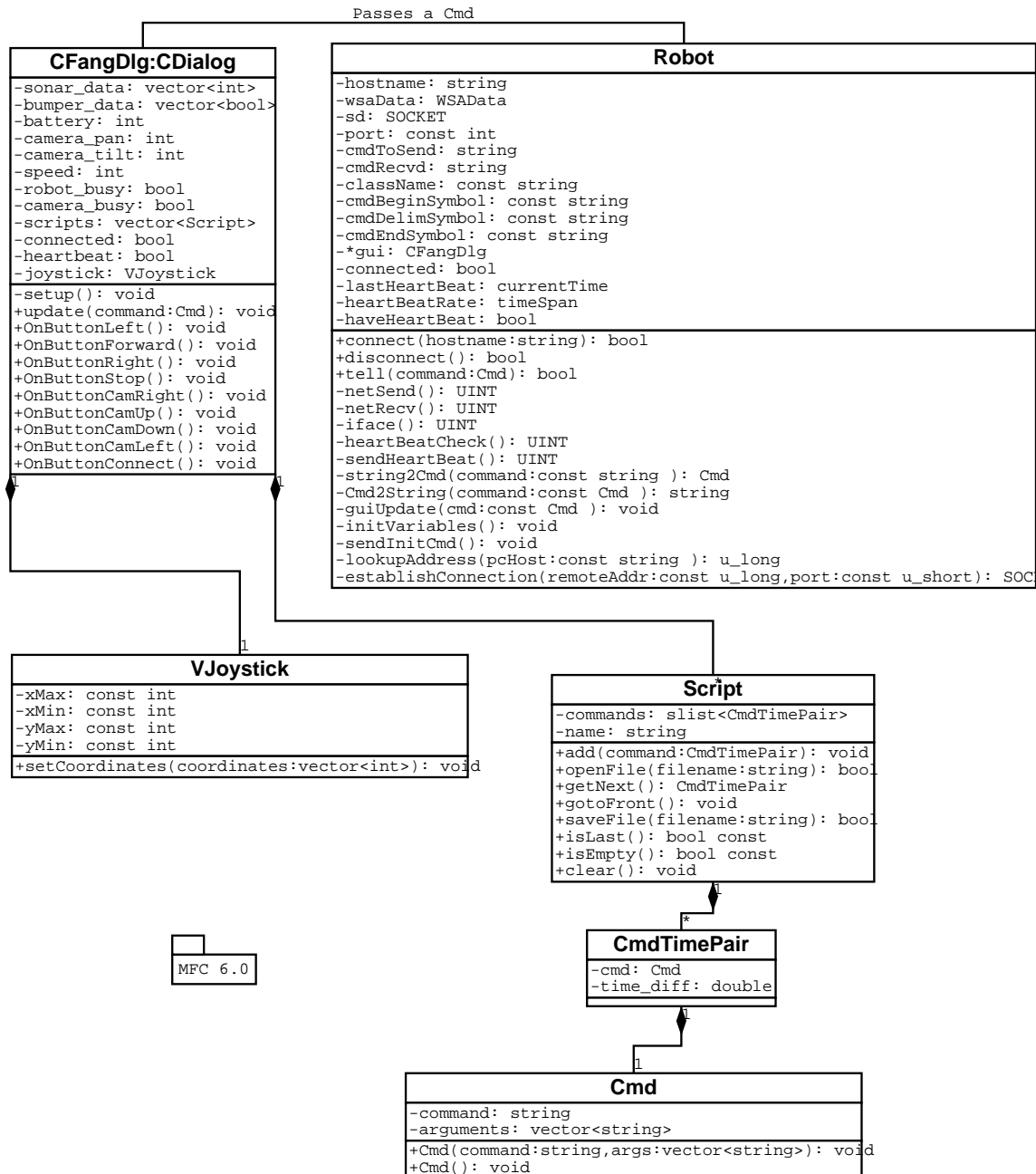


Figure 8.2: UML Diagram of FANG's static structure


```
// Use data member access functions, touching these directly
// is not thread safe.
bool connected;
currentTime lastHeartBeat;
timeSpan heartBeatRate;
bool haveHeartBeat;

// Accessed only through respective mutexes
// (used for inter-thread communication)
string cmdToSend;
string cmdRecvd;

CEvent sendEvent, sendEventDoneEvent, recvEvent, recvEventDoneEvent,
    netRecvEventDoneEvent,
    heartBeatCheckEvent, heartBeatCheckEventDoneEvent,
    sendHeartBeatEvent, sendHeartBeatEventDoneEvent,
    ifaceEvent, ifaceEventDoneEvent;
CMutex cmdToSendMutex, cmdRecvdMutex, lastHeartBeatMutex,
    heartBeatRateMutex, connectedMutex, haveHeartBeatMutex, guiMutex;

const string className;
const string cmdBeginSymbol;
const string cmdDelimSymbol;
const string cmdEndSymbol;

// Operations
public:
    // Sets the private data member *gui to the argument. Returns true
    // if gui was successfully stored.
    bool setGui(CFangDlg *gui);
    Robot();
    ~Robot();
    // Connects to the given robot server host. Returns true if it was able
    // to connect.
    bool connect(const string hostname);
    // Disconnects from the robot server, returns true it was able to disconnect
    bool disconnect();
    // Tells the robot to do command, returns true if it was able to send the
    // message to the robot.
    bool tell(const Cmd command);

    // These allow threads in member functions of this class to be started.
    // param is a pointer to this class instance's memory address.
    // Returns 0 on success.
```

```
friend UINT startNetSend(LPVOID param);
friend UINT startNetRecv(LPVOID param);
friend UINT startIface(LPVOID param);
friend UINT startHeartBeatCheck(LPVOID param);
friend UINT startSendHeartBeat(LPVOID param);
private:
    //
    // These are the threads of Robot.
    //
    // Sends data to the network
    UINT netSend();
    // Receives data from the network
    UINT netRecv();
    // Processes data received from the network
    UINT iface();
    // Checks that we are receiving a heartBeat from the robot server
    UINT heartBeatCheck();
    // Sends a heartBeat to the robot server
    UINT sendHeartBeat();

    //
    // Access functions for non-thread safe private data members
    //
    // get the heartBeatRate
    timeSpan getHeartBeatRate();
    // sets the heartbeat rate to newHeartBeatRate
    void setHeartBeatRate(const timeSpan newHeartBeatRate);
    // gets the time of the last heartBeat received
    currentTime getLastHeartBeat();
    // sets the time of the last heartBeat received
    void setLastHeartBeat(const currentTime newLastHeartBeat);
    // returns true if we have a heartBeat from the robot server
    bool getHaveHeartBeat();
    // sets whether or not we have a heartBeat from the server
    void setHaveHeartBeat(const bool haveIt);
    // returns true if we are currently connected to the robot server
    bool getConnected();
    // sets whether or not we are currently connected to the robot server
    void setConnected(const bool newConnected);

    // Parses command and turns it into a Cmd. Returns Cmd("", emptyVector)
    // if the command is invalid.
    Cmd string2Cmd(const string command) const;
    // Turns command into the format used to send the command over the network
```

```
    string Cmd2String(const Cmd command) const;

    // Passes cmd onto the gui
    void guiUpdate(const Cmd cmd);

    // Initializes variables
    void initVariables();
    // Sends the "init" command to the robot server
    void sendInitCmd();

    // if pcHost is a hostname, returns the ip address. if pcHost is an
    // ip address. returns pcHost
    u_long lookupAddress(const string pcHost) const;
    // brings up the connection to the remoteAddr (an ip addr) on the port
    // port. Returns the socket.
    SOCKET establishConnection(const u_long remoteAddr,const u_short port)const;
};

#endif
```

8.4.2 fang/cmd.h

```
// $Id: cmd.h,v 1.5 2002/03/27 02:28:28 ccf7f Exp $
// Flancrest Enterprises, Group 22
// About this file:
// The Cmd class allows code to talk about information that will be
// sent to/received from the robot server without knowledge of the
// actual protocol and data representation being necessary.

#ifndef CMD_H
#define CMD_H

#include <string>
#include <vector>
using namespace std;

class Cmd {
// Associations
// Attributes
private:
    string command;
    vector<string> arguments;
// Operations
public:
```

```

    Cmd();
    // commandName is the name of the command, args is a vector of the
    // arguments
    Cmd(string commandName, vector<string> args);
    // return the command
    string getCommand() const;
    // return the arguments
    vector<string> getArgs() const;
};

#endif

```

8.4.3 fang/gui.h

```

// $Id: gui.h,v 1.5 2002/03/04 19:28:06 ccf7f Exp $
// Flancrest Enterprises, Group 22

#ifndef GUI_H
#define GUI_H

#include "script.h"
#include "robot.h"
#include "cmd.h"
#include "vjoystick.h"

//The GUI class hides the response to user inputs and the way robot telemetry
//is stored. The GUI class is intended to handle all MFC events coming
//from the user and pass them along to the Robot class as Cmd structures.
class GUI {
    // Associations
    Robot * unnamed;
    VJoystick * unnamed;
    Script * unnamed;
    // Attributes
private:
    //the vector that stores the data received from the robots sonar
    //sensors
    vector<int> sonar_data;
    //the vector that stores the data received from the robots bump
    //sensors
    vector<bool> bumper_data;
    //the charge left in the robot's battery
    int battery;
    //the current pan angle of the camera, in degrees

```

```
int camera_pan;
    //the current tilt angle of the camera, in degrees
int camera_tilt;
    //the current speed setting of the robot
int speed;
    //indicates if the robot is performing an action or not
bool robot_busy;
    //indicates if the camera is performing an action or not
bool camera_busy;
    //stored scripts (sequences of robot commands)
vector<Script> scripts;
    //indicates if the program is connected to the remote host
bool connected;
    //indicates if the remote host is sending heartbeats at the
    // specified intervals
bool heartbeat;
    //the joystick that accepts user input
VJoystick joystick;
// Operations

public:
    vector<int> getSonar_data();
    void setSonar_data(vector<int> data);
    vector<bool> getBumper_data();
    void setBumper_data(vector<bool> data);
    int getBattery();
    void setBattery(int bat);
    int getCamera_pan();
    void setCamera_pan(int pan);
    int getCamera_tilt();
    void setCamera_tilt(int tilt);
    int getSpeed();
    void setSpeed(int speed);
    bool getRobot_busy();
    void setRobot_busy(bool busy);
    bool getCamera_busy();
    void setCamera_busy(bool busy);
    bool getConnected();
    void setConnected(bool con);
    bool getHeartbeat();
    void setHeartbeat(bool beat);

    //updates a part of the visual interface based on the command received
    void update ( Cmd command );
```

```
        //reacts to MFC events that control robot movement
void receiveMoveButtons ( buttonsData buttons );
        //reacts to MFC events that control camera movement
void receiveCameraSliders ( slidersData sliders );
        //reacts to MFC events that indicate the user wants to connect to the
        //remote host
void receiveConnect ( string hostname );
        //reacts to MFC events about various other parameters (speed, etc)
void receiveStatusInfo ( statusInfo status );
        //reacts to MFC events that affect scripting
void receiveScriptButtons ( buttonsData buttons );
private:
        //called to initialize an instance of the GUI class
        void setup ( );
};

#endif
```

8.4.4 fang/script.h

```
// $Id: script.h,v 1.6 2002/03/27 02:28:28 ccf7f Exp $
// Flancrest Enterprises, Group 22
// About this file:
// Script holds a series of command-time pairs. It can be used to automatically
// generate a series of predefined commands. It hides the storage and retrieval
// of script files.

#ifndef SCRIPT_H
#define SCRIPT_H

#include <string>
using namespace std;

#include "cmdtimepair.h"

class Script {
// Attributes
private:
        vector<CmdTimePair> commands;
        string name;
        int index;
// Operations
public:
```

```
// adds a CmdTimePair command to the end of the list of current commands
void add(CmdTimePair command);
// opens a file named filename that contains a stored list of commands
bool openFile(string filename);
// returns the current CmdTimePair and points to the next in the list
CmdTimePair getNext();
// makes the first CmdTimePair in the list the current CmdTimePair
void gotoFront();
// saves the list of CmdTimePairs to a file with the name filename
bool saveFile(string filename);
// returns true if the current CmdTimePair is the last in the list,
// false otherwise
bool const isLast();
// returns true if there are no CmdTimePairs in the list, false otherwise
bool const isEmpty();
// empties (erases) the list of CmdTimePairs
void clear();
};

#endif
```

8.4.5 fang/vjoystick.h

```
// $Id: vjoystick.h,v 1.6 2002/03/27 02:28:28 ccf7f Exp $
// Flancrest Enterprises, Group 22
// About this file:
// This class hides the way the joystick converts mouse movement
// into a set of commands destined for the robot.

#ifndef VJOYSTICK_H
#define VJOYSTICK_H

#include <vector>
using namespace std;

class VJoystick {
// Attributes
private:
    const int xMax;
    const int xMin;
    const int yMax;
    const int yMin;
// Operations
public:
```

```
    // Base constructor
    VJoystick();
    // sets the coordinates of the joystick
    // coordinates should be a 2 int vector containing <x-coordinate,
    // y-coordinate>, which represent the current relative coordinates
    // the joystick is pointing to, in pixels
    void setCoordinates(vector<int> coordinates);
};

#endif
```

8.4.6 fang/cmdtimepair.h

```
// $Id: cmdtimepair.h,v 1.6 2002/03/27 02:28:28 ccf7f Exp $
// Flancrest Enterprises, Group 22

// About this file:
// This class stores a command and a time difference to be used when
// scripting

#ifndef CMDTIMEPAIR_H
#define CMDTIMEPAIR_H

#include "cmd.h"

class CmdTimePair {
public:
    //Constructor
    CmdTimePair(void);
private:
    // Attributes
    Cmd cmd;
    double time_diff;
public:
    // Operations
    void setCmd(Cmd cmd);
    void setTime(double time_diff);
    Cmd getCmd();
    double getTime();
};

#endif
```


8.5 Use Cases

8.5.1 Connect to the Robot

See figure 8.3

- The user must first specify a hostname and click the connect button.
- The Robot Class will send a handshake to the robot server.
- Alternative Paths: If the handshake fails, alert the user.

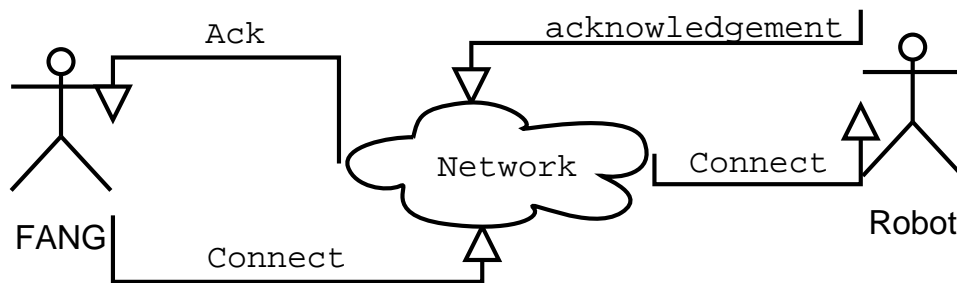


Figure 8.3: Use Case - Connect

8.5.2 Disconnect from the Robot

See figure 8.4

- The user must click the disconnect button.
- The Robot Class disconnects from the robot server.
- Precondition: The Robot Class must be connected to the network.

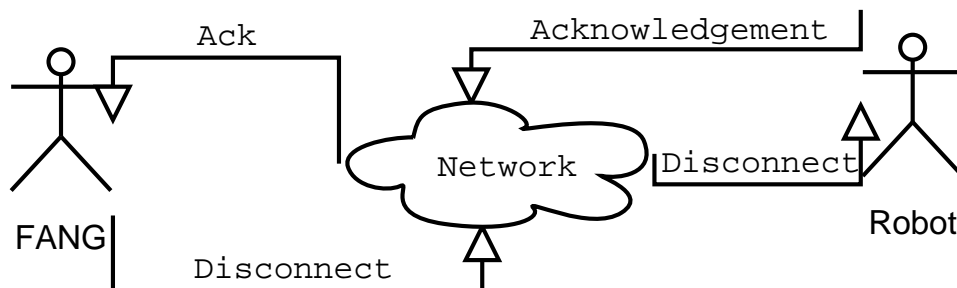


Figure 8.4: Use Case - Disconnect

8.5.3 Robot Movement

See figure 8.5

- The user clicks in the desired direction on the Virtual Joystick.
- The Robot Class sends the appropriate movement commands to the robot server.
- The robot server executes the desired movement commands.
- An acknowledgement of the completion of the command is sent back to FANG.
- Precondition: FANG must be connected to the network.
- Alternative Path: The acknowledgement is never completed, and the command does not finish. The user should be notified.
- Alternative Path: There is a loss of the heartbeat and the connection is broken. The user should be notified.

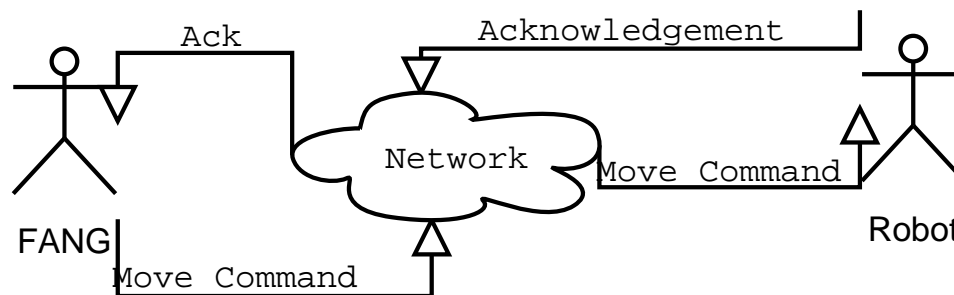


Figure 8.5: Use Case - Move Robot

8.5.4 Stop the Robot

See figure 8.6

- The user will simply press the stop button to stop the robot.
- The Robot Class sends the stop command to the robot server.
- The robot server executes the stop command.
- An acknowledgement of the completion of the command is sent back to FANG.
- Precondition: FANG must be connected to the network.

- Alternative Path: The acknowledgement is never completed, and the command does not finish. The user should be notified.
- Alternative Path: There is a loss of the heartbeat and the connection is broken. The user should be notified.

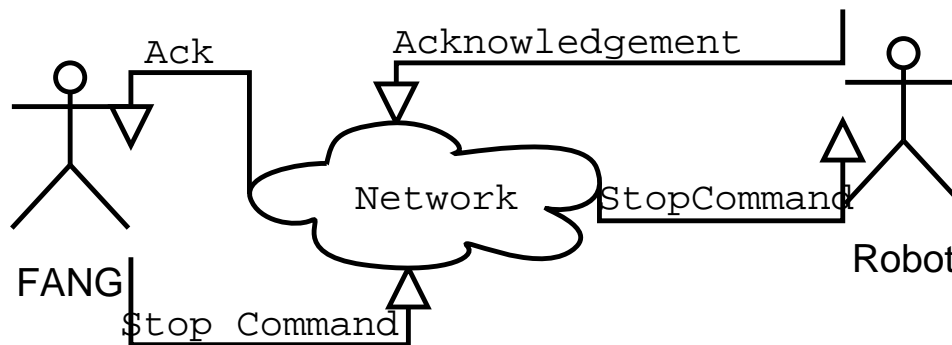


Figure 8.6: Use Case - Stop Robot

8.5.5 Turn the Robot while the Robot is stationary

See figure 8.7

- The user clicks and holds the rotate left or the rotate right button until the robot has been rotated to the desired direction.
- The Robot Class sends the appropriate rotate command to the robot server.
- The robot server executes the command.
- An acknowledgement of the completion of the command is sent back to FANG.
- Precondition: FANG must be connected to the network.
- Alternative Path: The acknowledgement is never completed, and the command does not finish. The user should be notified.
- Alternative Path: There is a loss of the heartbeat and the connection is broken. The user should be notified.

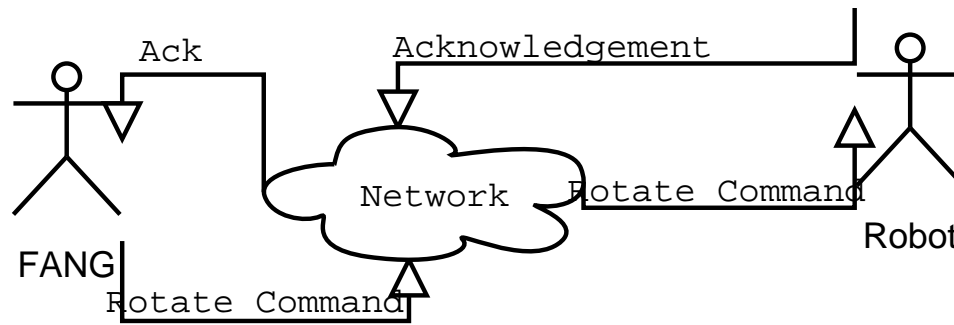


Figure 8.7: Use Case - Stationary Turn

8.5.6 Nudge the Robot in a desired direction

See figure 8.8

- The user clicks and releases on the forward or back button to nudge the robot forward or backward.
- The Robot Class sends the nudge command to the robot server.
- The robot server executes the nudge command, and sends an acknowledgement back to FANG.
- Precondition: FANG must be connected to the network.
- Alternative Path: The acknowledgement is never completed, and the command does not finish. The user should be notified.
- Alternative Path: There is a loss of the heartbeat and the connection is broken. The user should be notified.

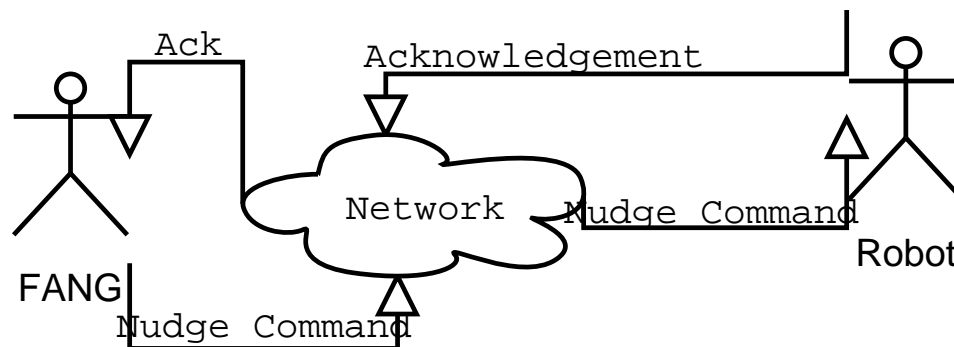


Figure 8.8: Use Case - Nudge Robot

8.5.7 Change the Speed of the Robot

See figure 8.9

- The user specifies a speed in the text box marked speed and click the update button.
- The Robot Server sends the appropriate command to the robot server.
- The robot server executes the command, and sends an acknowledgement back to FANG when the command is finished.
- Precondition: FANG must be connected to the network.
- Alternative Path: The acknowledgement is never completed, and the command does not finish. The user should be notified.
- Alternative Path: There is a loss of the heartbeat and the connection is broken. The user should be notified.
- Alternative Path: The speed is not valid. The robot should remain at current speed, and notify the user.

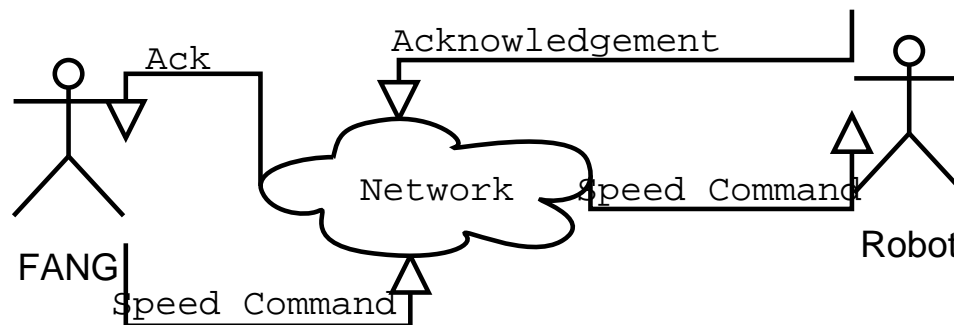


Figure 8.9: Use Case - Change Speed

8.5.8 Pan the Camera

See figure 8.10

- The user moves the slide bar the amount of pan he desires.
- FANG figures out the parameters and sends the pan command to the robot server.
- The robot server executes the command, and sends an acknowledgement back to FANG when the command is finished.

- Precondition: FANG must be connected to the network.
- Alternative Path: The acknowledgement is never completed, and the command does not finish. The user should be notified.
- Alternative Path: There is a loss of the heartbeat and the connection is broken. The user should be notified.

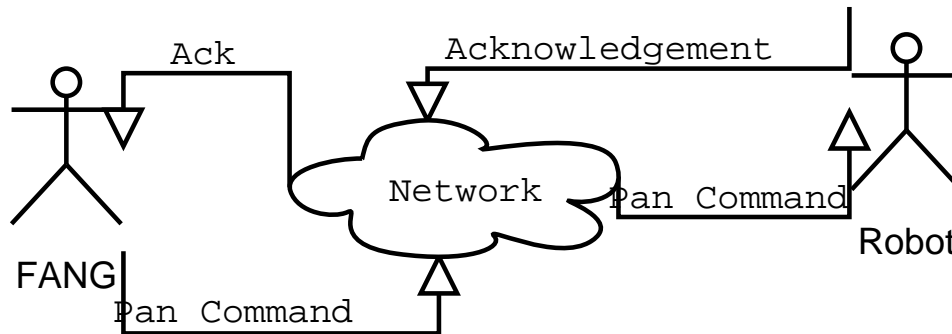


Figure 8.10: Use Case - Pan

8.5.9 Tilt the Camera

See figure 8.11

- The user moves the slide bar the amount of tilt he desires.
- FANG figures out the parameters and sends the tilt command to the robot server.
- The robot server executes the command, and sends an acknowledgement back to FANG when the command is finished.
- Precondition: FANG must be connected to the network.
- Alternative Path: The acknowledgement is never completed, and the command does not finish. The user should be notified.
- Alternative Path: There is a loss of the heartbeat and the connection is broken. The user should be notified.

8.6 Object Interactions

This section details the different interactions between the components of FANG. The interactions between the threads are shown in figure 8.12. The interactions between the objects and classes are shown in the sequence diagrams, which are in figures 8.13 through 8.15.

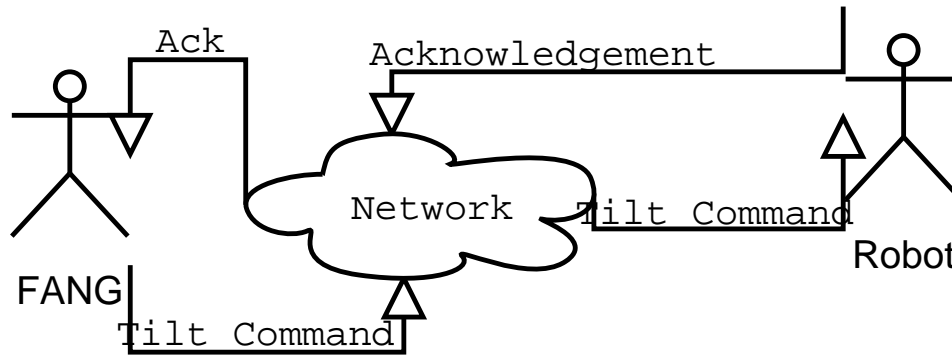


Figure 8.11: Use Case - Tilt

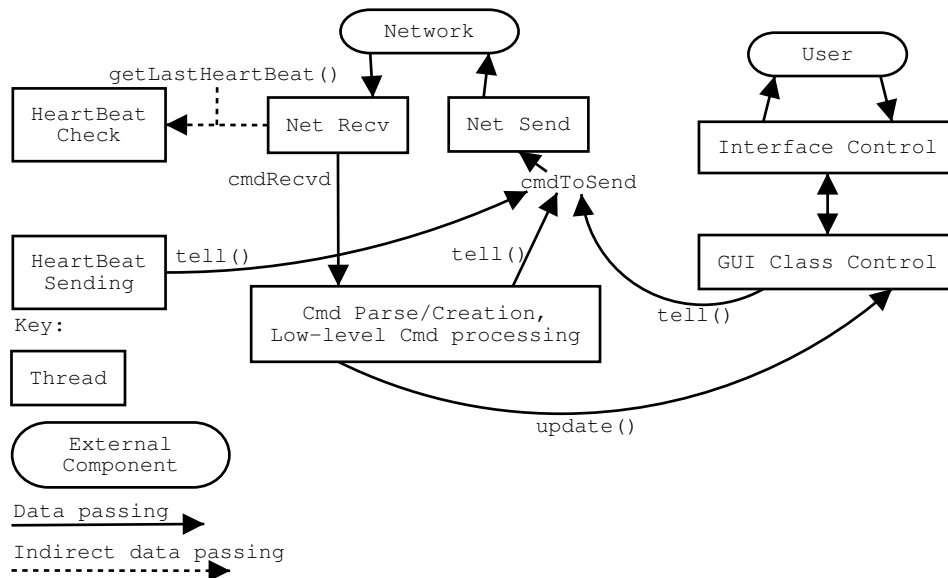


Figure 8.12: Thread Interactions

8.6.1 Sequence Diagrams

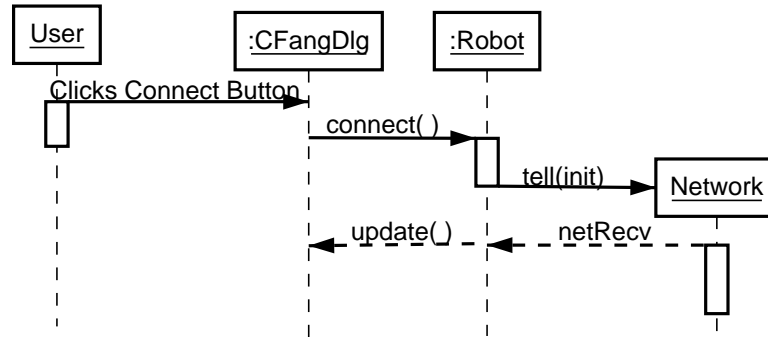


Figure 8.13: Sequence Diagram for Connecting to the On-board Software

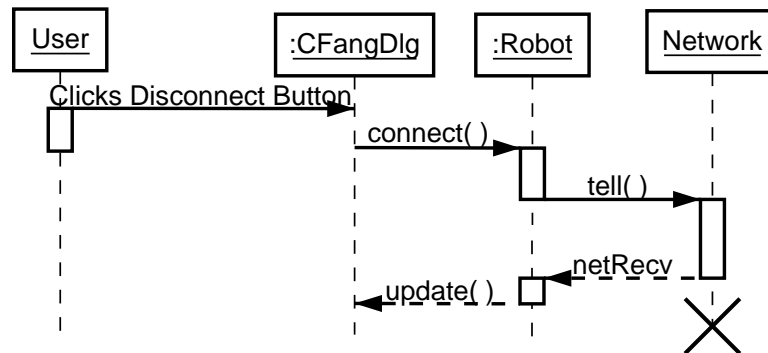


Figure 8.14: Sequence Diagram for Disconnecting from the On-board Software

8.7 Script File Format

The Script File Format, SFF, allows storage of scripts on a filesystem, which FANG can store and load. SFF is a new addition to FANG and is thus not detailed in [2], please see [6] for SFF's specification.

8.8 Possible Changes

- Number of cameras - Easy to change because it doesn't effect our Robot Class and the protocol is easily extendable
- Number of drive motors - This is more on the robot server side, and we can emulate this, therefore it can be easily changed
- Communications protocol (format, available commands) - Can be easily changed because a small number of functions deal with the actual commands

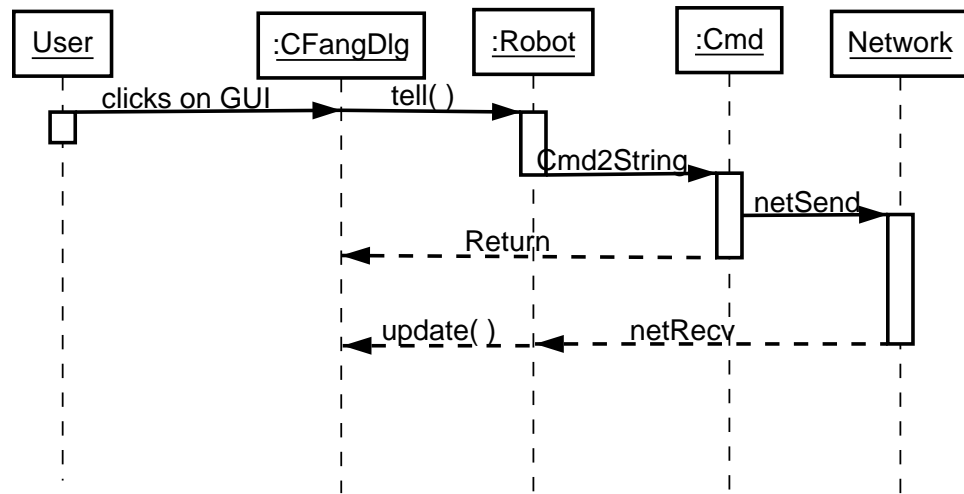


Figure 8.15: Sequence Diagram for Movement and Camera Commands

- Communication medium - Tell both groups immediately so appropriate changes can be made
- Specification - Update the new spec's info in the corresponding section in the design document, then update the code accordingly

8.9

Bibliography

- [1] Decepticon Industries, “Requivement Revisions,” Feb. 2002. <http://cs-tl3.cs.virginia.edu/cs340-21/documents.html>.
- [2] Decepticon Industries, “Final Specification,” Feb. 2002. <http://cs-tl3.cs.virginia.edu/cs340-21/documents.html>.
- [3] Flancrest Enterprises, “Mockup Requirements,” Feb. 2002. <http://cs-tl3.cs.virginia.edu/cs340-22/docs/materials/>.
- [4] Flancrest Enterprises, “Robot Server Requirements Specification,” Feb. 2002. <http://cs-tl3.cs.virginia.edu/cs340-22/docs/materials/>.
- [5] Decepticon Industries and Flancrest Enterprises, “Communication Protocol Specification,” Mar. 2002. <http://cs-tl3.cs.virginia.edu/cs340-22/docs/materials/>.
- [6] Flancrest Enterprises, “Script File Format Specification,” Mar. 2002. (See us for this document).

Chapter 9

Testing Report

Postlab 8
April 3, 2002

9.1 Goals for Our Prototype

- Connect to and disconnect from Decepticon's robot server
- From the GUI, be able to:
 - move robot forward
 - rotate robot
 - move camera
- Multithread the Robot class (recv, send, heartBeatSend, heartBeatCheck, and low-level-command processing)
- Make use of these commands from our protocol:
 - pr
 - sp
 - reset
 - st
 - bumpCamera
 - init
 - disconnect
 - ack
 - hearBeat
 - unknownCmd
 - invalidArgs

9.2 Milestones Set for This Week

Milestones (note that each of these used a sign off sheet to mark completion):

- Review design: Wed, all. Completed.
- Decide on goals: Wed, both groups. Completed.
- Tell Scott which commands to implement in FURL: Wed. Completed.
- Setup base code for FANG: Thur, Chris and Emilio. Completed.
- Individual coding: Fri – Sat, Chris and Emilio. Completed.
- Test with FURL: Sat, Chris and Emilio. Completed.
- Test with robot server: Sunday, Chris and partner group. Completed, Wednesday.

9.3 Testing Activity

9.3.1 Testing with FURL

What	When	Outcome	By Whom
Connect w/ init message	Fri	No problems	Chris
Recv and send data using Robot class	Sat	Not killing recv thread at disconnect. Will fix before next lab.	Chris
Send and recv heartBeats	Sat	Doesn't allow for heartBeat of zero. Will fix before next lab.	Chris
GUI sends correct commands to move robot forward, rotate robot, and move camera	Sun	No problems	Emilio
unknownCmd	Sat	No problems	Chris
invalidArgs	Sat	No problems	Chris

9.3.2 Testing with robot server

Testing done Wednesday 12am – 2am, by Chris and Decepticon.

What	Outcome
------	---------

Connect w/ init message	Server doesn't use init message. Temporary solution: FANG sends individual commands corresponding to what init would do.
GUI controls move robot forward, rotate robot, and move camera	Can only move camera to specific positions, can't bumpCamera. vm not implemented on server side, switched to using pr and sp for now.
heartBeat	Server does not yet support.
unknownCmd	Server does not yet support.
invalidArgs	Server does not yet support.
reset	Ignored by server, stopOnBump not yet implemented.

9.4 FANG's Source

FANG's source code is in a separate document, see <http://cs-tl3.cs.virginia.edu/cs340-22/>

Chapter 10

System Performability

Postlab 9
April 8, 2002

10.1 Performability Goals

- **Goal:** User can move robot and change speed using game pad
Result: Success; game pad allows user to change movement speed, move forward and backward, rotate, and rotate while moving forward or backward.
- **Goal:** User can move camera and robot using GUI
Result: Success; GUI allows user to set absolute camera direction or increment direction up, down, left, or right by a small angle. GUI allows user to move robot forward, backward, left, right, or a combination of these, with adjustable speed, and in either “run” or “bump” modes.
- **Goal:** Control of robot is intuitive — new users can learn quickly
Result: Success; Yannick Lotière learned to drive the robot using our game pad in less than a minute. Controlling the robot is similar to playing a video game. Keyboard reading not yet implemented, scheduled for next week.
- **Goal:** Smooth movement of camera and robot
Result: Success, though we plan to use an analog game pad in the future because the digital pad has only three possible states for each axis. An analog stick will allow smoother transition along the translation/rotation gradient.
- **Goal:** Software stability
Result: Partial success; only one known bug: fang hangs if a disconnect is attempted and the robot server has crashed. This is scheduled to be corrected next week.
- **Goal:** Display sonar and bumper data
Result: Partially implemented; scheduled for completion next week

10.2 Milestones

Each milestone is checked off on our “Current Week’s Milestones” document upon completion.

What	Due	Who
Prototype joystick interfacing, able to query axes and buttons	3/28	Chris
Do not expect/send heartBeat when rate is zero	3/30	Chris
iface thread exits upon disconnect()	3/30	Chris
connect() waits for ack	3/30	Chris
connect()/disconnect() check connect state first	3/30	Chris
Robot only tells gui the first time we lose the heartBeat	3/30	Chris
Robot updates its heartBeatRate if the gui sends a setHeartBeatRate	3/30	Chris
ws-util.cpp uses fangDebug() rather than cerr	3/30	Chris
Display sonar and bumper data	3/30	Emilio
Add additional movement buttons	3/30	Emilio
Write joystick class	3/31	Chris
Add commands being implemented this week in FANG to FURL	3/31	Tommy
Move FURL from using char* to string and char** to vector<string>	4/1	Chris and Tommy
GUI uses joystick for robot movement and speed control	4/2	Emilio

10.3 Integration Testing

Date: 4/2

Time: 1900 – 2130

Location: Olsson 001

Group members present: Chris Frost, Emilio Lahr-Vivaz

Partner group members present: Keen Browne, Phil Harton

Goals and results (from our side):

- **Goal:** Robot movement commands work: vm, pr, sp, st
Result: Success
- **Goal:** init and disconnect commands work
Result: Success
- **Goal:** Status commands getBp and getSn work
Result: Success

- **Goal:** heartBeat feature works
Result: Success
- **Goal:** unknownCmd and invalidArgs commands work
Result: Success; we log via our debug console
- **Goal:** moveCamera and bumpCamera commands work
Result: Success
- **Goal:** Read from joystick and control robot's movement
Result: Success

10.4 Implementation Status

- **Function:** Connect and disconnect from robot server.
Status: Connect implemented, tested, and working. Disconnect implemented, but does not pass test for disconnecting from crashed robot server.
- **Function:** Rotate robot in place both left and right.
Status: Implemented, tested, and working from GUI buttons and through joystick.
- **Function:** Move forward and backward.
Status: Implemented, tested, and working from GUI buttons and through joystick.
- **Function:** Moving forward and backward while turning.
Status: Implemented, tested, and working from GUI buttons and through joystick.
- **Function:** Stopping robot while moving.
Status: Implemented, tested, and working from GUI buttons and through joystick
- **Function:** Adjusting speed of robot movements.
Status: Implemented, tested, and working from GUI buttons and through joystick.
- **Function:** Ability to control movement by speed or by distance to travel.
Status: Implemented, tested, and working through GUI.
- **Function:** Ability to move camera to an absolute position.
Status: Implemented but not tested through GUI sliders. Implementation not started for joystick control.
- **Function:** Ability to move camera by a fixed angle.
Status: Implemented but not tested on GUI through buttons.

- **Function:** Ability to receive and display sonar data from robot.
Status: Implemented but not tested as visual display on GUI.
- **Function:** Ability to receive and display bumper data from robot.
Status: Implemented, tested, and working as visual display on GUI.
- **Function:** Ability to read input from keyboard.
Status: Not implemented.

Chapter 11

Implementation State

Postlab 10
April 22, 2002

11.1 Implementation Functionality

- **Function:** Connect and disconnect from robot server.
Status: Connect and disconnect implemented, tested, and working.
- **Function:** Rotate robot in place both left and right.
Status: Implemented, tested, and working from GUI buttons, keyboard, and through joystick.
- **Function:** Move forward and backward.
Status: Implemented, tested, and working from GUI buttons, keyboard, and through joystick.
- **Function:** Moving forward and backward while turning.
Status: Implemented, tested, and working from GUI buttons, keyboard, and through joystick.
- **Function:** Stopping robot while moving.
Status: Implemented, tested, and working from GUI buttons, keyboard, and through joystick
- **Function:** Adjusting speed of robot movements.
Status: Implemented, tested, and working from GUI buttons, keyboard, and through joystick.
- **Function:** Ability to control movement by speed or by distance to travel.
Status: Implemented, tested, and working through GUI.
- **Function:** Ability to move camera to an absolute position.
Status: Implemented, tested, and working through GUI sliders and joystick..

- **Function:** Ability to move camera by a fixed angle.
Status: Implemented and testing on GUI through buttons and joystick.
- **Function:** Ability to receive and display sonar data from robot.
Status: Implemented, tested, and working as visual display on GUI.
- **Function:** Ability to receive and display bumper data from robot.
Status: Implemented, tested, and working as visual display on GUI.
- **Function:** Ability to read input from keyboard.
Status: Implemented, tested, and working.
- **Function:** Ability to set rates for periodic data and heartBeat.
Status: Implemented, tested, and working.

11.2 Final Design vs Design Document

11.2.1 Not Present in Final Design

We decided we didn't need a virtual joystick since we have an actual joystick, and so the vjoystick class was eliminated.

Now knowing that we can not script for dancing, we didn't have any use for scripting capabilities, and so removed the script and cmdtimepair classes.

11.2.2 New in Final Design

We have added the ability to use a joystick, and thus now have a Joystick class. We added a configuration dialog through the CConfigDlg class. And we added logging capabilities for debugging and error messages through the Errors class.

11.2.3 Changed in Final Design

CxFangDlg

- Added parameters for configuration dialog
- Added many const parameters
- More mfc events (eg sliders)
- Added joystick interface capabilities
- Joystick-polling thread added

Robot

- Ability to turn logging on/off

11.3 Changes Since Enhanced Prototype

See Section 11.5, page 67.

11.4 Key Implementation Parameters

11.4.1 Lines of Code

fangDlg.cpp	1179
robot.cpp	840
*.cpp	2586
Total (*.cpp + *.h)	3210

11.4.2 Misc

Number of Classes: 9

Number of Implementation Files: 10

Average Number of Lines per Implementation File: 258.6

Number of CVS Commits: 77

11.5 ChangeLog

2002-04-09 e19d <e19d@cs-tl3.cs.virginia.edu>

* fang/fang.plg, fang/fangDlg.cpp:
fixed joystick polling for buttons and digital pad

* fang/fang.plg, fang/fangDlg.cpp: added joystick functionality -
speed control, reset after bump, reset robot and camera
TODO: digital movement implemented but joystick class needs work

* fang/fang.plg, fang/fangDlg.cpp: streamlined some code
tied user input refresh rates to actual refresh rates sent

* fang/Resource.h, fang/fang.aps, fang/fang.dsp, fang/fang.plg, fang/fang.rc, fan
changed bump switches to display which switch is hit
slightly modified bitmaps (weren't symmetrical before)

2002-04-07 e19d <e19d@cs-tl3.cs.virginia.edu>

* fang/fang.aps, fang/fangDlg.cpp:
fixed bump rotate (used to only rotate one direction)

2002-04-06 ccf7f <ccf7f@cs-tl3.cs.virginia.edu>

- * fang/ConfigDlg.cpp, fang/ConfigDlg.h, fang/Resource.h, fang/errors.cpp, fang/fangDlg.h:
- CFangDlg:
- reversed axis for camera joystick
- fixed magnitude of joystick axis
- no longer send multiples of a non-affecting command
- turn left and right work in bump and run modes
- option to log comm i/o

2002-04-06 el9d <el9d@cs-tl3.cs.virginia.edu>

- * fang/ConfigDlg.cpp, fang/fang.aps, fang/fang.plg, fang/fangDlg.cpp:
- linked debug window check box to go through CFangDlg instead of directly to robot

2002-04-06 ccf7f <ccf7f@cs-tl3.cs.virginia.edu>

- * TODO, fang/fang.plg, fang/robot.cpp:
- If we don't have a heartBeat while disconnecting, we now don't wait for an (ack disconnect) since the server may not be responding.
- * TODO, fang/fang.plg, fang/robot.cpp, fang/robot.h:
- Can change whether or not we log communication i/o, thread entering/exiting to fangDebug().
- * fang/ConfigDlg.cpp, fang/errors.cpp, fang/errors.h, fang/fang.plg:
- Added Errors::ShowConsole() to show/hide debug console.

2002-04-05 el9d <el9d@cs-tl3.cs.virginia.edu>

- * fang/fang.aps, fang/fang.clw, fang/fang.plg, fang/fang.rc:
- modified text in about box
- * fang/fang.aps, fang/fang.plg, fang/fang.rc, fang/fangDlg.cpp, fang/fangDlg.h:
- modified keyboard input so that it only sends a command once if the key is held d
- * fang/fang.aps, fang/fang.clw, fang/fang.plg, fang/fang.rc, fang/res/Thumbs.db,
- changed icon
- * fang/fang.dsp, fang/fang.plg, fang/fangDlg.h:
- really removed vjoystick, script, cmdtimepair and all references to them
- * fang/cmdtimepair.cpp, fang/cmdtimepair.h, fang/res/Thumbs.db, fang/script.cpp,
- trying again to remove cmdtimepair, vjoystick, and script

- * fang/fang.plg, fang/fangDlg.cpp: tied user config data to bump distances

- * fang/ConfigDlg.cpp, fang/ConfigDlg.h, fang/Resource.h, fang/fang.aps, fang/fang
added config dialog box
added ConfigDlg.h and .cpp
added member variables to CFangDlg to control config parameters
added mutators and getters for those member variables
TODO: need to tie member variables into actual commands

- * fang/fang.plg, fang/fangDlg.cpp, fang/fangDlg.h:
added key support for movement, camera, speed control

2002-04-04 el9d <el9d@cs-tl3.cs.virginia.edu>

- * fang/fangDlg.h, fang/fang.aps, fang/fang.clw, fang/fang.plg, fang/fang.rc, fang
changed joystick to handle dual analog (move / camera)

2002-04-04 ccf7f <ccf7f@cs-tl3.cs.virginia.edu>

- * Makefile: Updated list of files for new joystick.[h|cpp] location.

- * TODO, fang/fang.plg, fang/joystick.cpp, fang/joystick.h:
Moved from using dynamic memory for the private dm joyinfo to static.
(Using dynamic memory is a relic from avoiding including windows.h in the
header file.)

- * fang/fang.dsp, fang/fang.plg, fang/joystick.cpp, fang/joystick.h:
 - Moved joystick class from a seperate project into the fang project
(with mfc, dinput.h wasn't including the right files. But I found
the corrected header).
 - Shortened a couple lines that were way over 80 chars long

2002-04-03 el9d <el9d@cs-tl3.cs.virginia.edu>

- * fang/fang.plg, fang/fangDlg.cpp, fang/fangDlg.h:
rescaled sonar, should only reach to about 3 feet now
lined up output sonar display to actual robot sonars

- * fang/fangDlg.cpp, fang/fang.plg:
lined up sonar drawing with actual sonars

2002-04-03 ccf7f <ccf7f@cs-tl3.cs.virginia.edu>

- * TODO: Added a couple Robot and Joystick todo items

2002-04-03 e19d <e19d@cs-tl3.cs.virginia.edu>

- * fang/fang.plg, fang/fangDlg.cpp:
removed some debugging stuff from last version
fixed initial cursor so it is only initial, not every time paint is called

- * fang/fang.aps, fang/fang.plg, fang/fangDlg.cpp: fixed redrawing problems
made cursor initially active in connect text box

Chapter 12

Example Management Report

Postlab 1
January 30, 2002

12.1 Management Responsibilities

Person	Responsibility
Chris	Configuration management and file system control
Emilo	Presentation preparation
Scott	Web development
Thomas	Scheduling and task assignment
Chris and Scott	Document Preparation

12.2 Previous Responsibilities

Chris	
Status	Task
Completed 1/29	Initial website work
Completed 1/29	Write template for documents and management report
Completed 1/30	Investigate sockets

Emilio	
Status	Task
Completed 1/29	Investigate MFC

Scott	
Status	Task
Completed 1/30	Investigate robot movement and sensors

Thomas	
Status	Task
Completed 1/29	Investigate MFC

12.3 Meeting Reports

12.3.1 Meeting #1

- Date: 1/27
- Time: 11:00 – 13:00
- Location: Olsson 001
- Attendees: Chris Frost, Emilio Lahr-Vivaz, Scott Rein, and Thomas Whanger

Agenda:

- Meet eachother
- Choose areas of expertise
- Review and split-up postlab assignments

Results:

- Chose “Flancrest Enterprises” as our name
- Accomplished all items on the agenda (documented on our website)
- Chose Wednesdays at 12:00 in Olsson 001 as our weekly meeting time and location

12.3.2 Meeting #2

- Date: 1/29
- Time: 19:00 – 20:00
- Location: Olsson 001
- Attendees: Chris Frost, Emilio Lahr-Vivaz, Scott Rein, and Thomas Whanger

Agenda:

- Split-up work for coming week
- Put together document preparation process
- Checkup on individual efforts

Results:

- Accomplished all items on agenda

12.4 Activity Log

Chris		
Date	Time	Purpose
1/27	11:00 – 12:30	Initial meeting with group, decided how to attack postlab one and prelab two.
1/27	12:30 – 14:30	L ^A T _E X macro writing for use in our documents.
1/28	14:00 – 15:00	Sockets
1/28	20:30 – 23:30	Website work, document construction, and team list setup
1/29	19:00 – 20:00	Second group meeting
1/29	21:30 – 23:30	Management report, writeup document generation process, and typing
1/30	9:00 – 10:00	Sockets

Emilio		
Date	Time	Purpose
1/27	11:00 – 12:30	Initial meeting with group, decided how to attack postlab one and prelab two.
1/27	12:30 – 13:00	Worked on prototype for MFC interface.
1/29	19:00 – 20:00	Second group meeting

Scott		
Date	Time	Purpose
1/27	11:00 – 12:30	Initial meeting with group, decided how to attack postlab one and prelab two.
1/29	19:00 – 20:00	Second group meeting, learned eXceed interface with Chris
1/29	21:00 – 22:00	Prototype to move robot
1/30	12:30 – 13:00	Prototype to get state of robot

Thomas		
Date	Time	Purpose
1/27	11:00 – 12:30	Initial meeting with group, decided how to attack postlab one and prelab two.
1/27	12:30 – 13:00	Worked on prototype for MFC interface.
1/29	19:30 – 20:00	Second group meeting, assigned the rest of the assignments, wrote prototype document.

12.5 Schedule of Work Assignments

Person(s)	Date Due	Assignment
Chris	2/3 meeting	Develop template for system specification document
Emilio	2/3 meeting	Document risks
Scott	2/3 meeting	Management report
Thomas	2/3 meeting	Coordinate team meeting
Emilio, Scott, and Thomas	2/3 meeting	Debug VisualBasic mockup
Emilio	2/3	Presentation

Projects are done when:

- SSD template: all methods of expressing data we have thought of have macros for them and group has ok'd.
- Risk documentation: Risks that each person has thought of have been written up and group has ok'd.
- Management report: All members have submitted their sections and group has ok'd.
- Coordinate team meeting: Meeting time has been agreed to by both teams.
- Debug VB mockup: Emilio, Scott, and Thomas agree that all relevant data visualization types have been mocked up.
- Presentation: Documentation of our major work has been written up and group has ok'd.

12.6 Unresolved Problems

- No known unresolved problems.

Chapter 13

Example Management Report

Postlab 3
February 11, 2002

13.1 Management Responsibilities

Person	Responsibility
Chris	Configuration management and file system control
Emilio	Presentation preparation
Scott	Web development
Thomas	Scheduling and task assignment
Chris and Scott	Document Preparation

13.2 Previous Postlab Responsibilities

Chris	
Status	Task
Completed 2/10	Communications protocol
Completed 2/10	Robot requirement specifications

Emilio	
Status	Task
Completed 2/10	Debugger requirements specification
Completed 2/10	Debugger specifications draft
Completed 2/10	Updated debugger mock-up
Completed 2/10	Powerpoint presentation

Scott	
Status	Task
Completed 2/11	Robot requirements specification
Completed 2/11	Website revisions
Completed 2/11	Management report

Tommy	
Status	Task
Completed 2/10	Debugger requirements specification
Completed 2/10	Debugger specifications draft
Completed 2/10	Updated debugger mock-up

13.3 Meeting Reports

13.3.1 Meeting #1

- Date: 2/8
- Time: 1100 – 1200
- Location: Stacks
- Attendees: Chris Frost, Emilio Lahr-Vivaz, and Decepticon Industries

Agenda:

- Develop robot server requirements
- Further develop gui requirements
- Further develop debugger requirements
- Appoint group contacts for communications protocol spec work

Results:

- All agenda items discussed
- Keen and Chris are the group contacts for the communications protocol spec work

13.3.2 Meeting #2

- Date: 2/10
- Time: 1430 – 1800
- Location: Olsson 001

- Attendees: Chris Frost, Emilio Lahr-Vivaz, Scott Rein, and Thomas Whanger

Agenda:

- Critique on debugger interface
- Discussion of debugger and robot specs
- Develop Powerpoint presentation

Results:

- All agenda items discussed
- Tommy and Emilio will complete debugger specs
- Chris and Scott will complete robot specs
- Powerpoint presentation will be emailed to group for confirmation

13.4 Activity Log

Chris		
Date	Time	Purpose
2/6	1430 – 1500	Updated L ^A T _E X templates in response to TA feedback
2/8	1100 – 1200	Group 21 and 22 meeting
2/8	1600 – 1700	Worked on comm protocol with Keen
2/8	2200 – 0030	Worked on beginnings of requirements spec
2/9	1300 – 1500	Worked on comm protocol with Keen
2/9	1500 – 1815	User Interface requirements and requirements specification work
2/10	0715 – 0900	Protocol specification
2/10	0930 – 1000	L ^A T _E X macros and image use research
2/10	1000 – 1230	Robot server requirements spec with Scott
2/10	1430 – 1800	Req specs for robot and debugger, presentation, and debugger specification with group
2/10	2130 – 0145	Robot server req specs, cvs help, document builds

Emilio		
Date	Time	Purpose
2/8	1100–1200	Double group meeting, went over requirements and communication protocol
2/9	1400–1500	Revised Debugger
2/10	1300–1900	Changes to Debugger, Requirements Specification
2/10	2100–0130	Changes to Requirements Specification, Draft of Specification, and worked on presentation

Scott		
Date	Time	Purpose
2/10	1000 – 1230	Robot server requirements spec with Scott
2/10	1600 – 1730	Req specs for robot and debugger, presentation, and debugger specification with group
2/10	1130 – 0030	Website revision with TA comments
2/11	1230 – 0130	Robot specification
2/11	0130 – 0230	Document management
2/11	1215 – 1300	Website maintenance: posting new files, management report

Thomas		
Date	Time	Purpose
2/9	1400–1500	Revised Debugger
2/9	1600–1700	Chris Meeting
2/10	1300–1900	Changes to Debugger, Requirements Specification, and Draft of Specification
2/10	2100–0130	Changes to Debugger, Requirements Specification, and Draft of Specification

13.5 Schedule of Work Assignments

Person(s)	Date Due	Assignment
All	2/17	Final versions of requirement specs
Chris and Scott	2/17	Inspection of final specification
Tommy and Emilio	2/17	Final version of the specification document
Chris and Decepticon Industries	2/17	Document draft protocol specification
Scott	2/18	Management report
All	2/17	Implementation planning: cost estimate
All	2/17	– Milestones
All	2/17	– Detailed schedule
All	2/17	– Work breakdown
Chris and Scott	2/17	– Integration into Pert and Gantt charts
All	2/17	– Process report
All	2/17	– Critique of Decepticon's specification
Emilio	2/17	Powerpoint presentation

13.6 Unresolved Problems

- All problems resolved.

Chapter 14

Example Management Report

Postlab 6
March 4, 2002

14.1 Management Responsibilities

Person	Responsibility
Chris	Configuration management and file system control
Emilio	Presentation preparation
Scott	Web development
Thomas	Scheduling and task assignment
Chris and Scott	Document Preparation

14.2 Previous Responsibilities

Chris	
Status	Task
3/3	Static structure uml diagram
3/3	Design document portions
3/3	Final version of the protocol specification

Emilio	
Status	Task
3/03	Completed UML
3/03	Completed c++ class interfaces
3/04	Completed presentation for lab 7

Scott	
Status	Task
3/03	Completed UML
3/03	Completed c++ class interfaces
3/04	Completed presentation for lab 7

Thomas	
Status	Task
3/03	Completed UML
3/03	Completed c++ class interfaces
3/04	Completed presentation for lab 7

14.3 Meeting Reports

14.3.1 Meeting #1

- Date: 03/03
- Time: 1230 – 1245
- Location: Olsson 001
- Attendees: Chris Frost, Emilio Lahr-Vivaz, Scott Rein, and Thomas Whanger

Agenda:

- See if anyone has questions on previous assignments.
- Schedule rest of tasks that need to be done.
- Make sure Thomas is caught up after being away for a week.

Results:

- All agenda items were discussed.
- Questions were addressed.
- Thomas will do the Object Interactions and Use Cases.
- Chris will do the Implementation Document.
- Thomas learned about the proposed design.

14.4 Activity Log

Chris		
Date	Time	Purpose
2/28	1800 – 1930	Worked with Emilio on FANG’s design.
3/2	1100 – 1400	Worked with Emilio on FANG’s design.
3/2	1600 – 1800	Design doc and FANG code basic setups.
3/2	2015 – 2345	Worked on the design doc.
3/3	0900 – 1015	Worked on the design doc.
3/3	1100 – 1230	Explained support tool design to Scott and FANG’s design to Tommy.
3/3	1230 – 1245	Group meeting.
3/3	1415 – 1600	Protocol spec work.
3/3	2000 – 2300	Design document work.
3/3	2315 – 0115	Protocol spec work.

Emilio		
Date	Time	Purpose
2/28	1800 – 1930	started UML using Dia
3/02	1100 – 1445	worked on UML using Dia
3/03	1230 – 1500	worked on UML, c++ interfaces
3/03	2000 – 2100	finished UML document after getting help
3/04	1100 – 1130	worked on presentation

Scott		
Date	Time	Purpose
3/3	1100-1400	Research on sockets, beginning coding of tool
3/4	1000-1300	Completion of tool

Thomas		
Date	Time	Purpose
3/3	1130 – 1300	Met with Chris to find out about what happened while I was away, Meeting
3/3	1900 – 2300	Object Interactions and Use Cases

14.5 Unresolved Problems

- FANG’s specification is not complete

Chapter 15

Example Management Report

Postlab 9
April 3, 2002

15.1 Management Responsibilities

Person	Responsibility
Chris	Configuration management and file system control
Emilio	Presentation preparation
Scott	Web development
Thomas	Scheduling and task assignment
Chris and Scott	Document Preparation

15.2 Previous Responsibilities

Chris	
Status	Task
Completed 3/29	Enhance Robot for second prototype
Completed 4/2	Testing with FURL and Decepticon
Completed 3/30	joystick prototyping
Completed 4/3	Write testing report

Emilio	
Status	Task
4/2	Worked on GUI interface (joystick functions, displaying data, adding capabilities)

Scott	
Status	Task
Completed 4/3	Added Perl scripting to website

Thomas	
Status	Task
Completed 3/31	Expand the support tool
Completed 4/01	Update risk and schedule on website

15.3 Meeting Reports

15.3.1 Meeting #1

- Date: 03/29
- Time: 1300 – 1330
- Location: Thorton Stacks
- Attendees: Chris Frost, Emilio Lahr-Vivaz, Scott Rein, and Thomas Whanger

Agenda:

- Discuss and develop goals for the enhanced prototype
- Develop milestones and schedule for the enhanced prototype
- Divide up the work amongst the group
- Find date and place for integration testing

Results:

- All agenda items were discussed
- The goals for the enhanced prototype were developed
- The schedule and milestones were also developed
- The work was divided among Chris, Emilio, and Thomas
- Integration testing will be at 5:30 Monday at Olsson 001

15.4 Activity Log

Chris		
Date	Time	Purpose
3/29	1145 – 1200	Joystick prototyping
3/29	1430 – 1500	Robot class correcting
3/29	1545 – 1900	Robot class correcting
3/29	2045 – 0245	Robot class work and FURL development and debugging
3/30	1230 – 1300	Worked with Emilio on the joystick
3/30	1400 – 1430	Added ability to create source code listings in FANG's Makefile
3/30	2330 – 0030	Helped Emilio with Unix → Win → Unix CVS issues
3/31	1045 – 1230	Working with Emilio, got wincvs working
4/1	0730 – 0830	Began testing report for last week's tests, published preliminary copies of this and the mngmnt report
4/1	1030 – 1130	Worked on FURL
4/1	1530 – 1730	Joystick integration into FANG with Emilio
4/1	1730 – 1800	Testing with Decepticon
4/2	1730 – 1900	Direction discussion with Keen, netRecv debugging
4/2	1900 – 2100	Testing with Decepticon
4/3	1000 – 1045	postlab 8 testing writeup

Emilio		
Date	Time	Purpose
3/29	1600 – 2000	*tried* to get keyboard linking working
3/30	1200 – 1600	implemented visual displaying of data in GUI, implemented more functionality
4/1	1030 – 1200	implemented more of GUI
4/2	1500 – 1600	implemented more of GUI
4/2	1900 – 2130	tested robot functionality with partner group, debugged and changed a few errors in GUI, practiced robot control.

Scott		
Date	Time	Purpose
4/3	1300 – 1330	Management report
4/3	1330 – 1400	Perl scripting for website

Thomas		
Date	Time	Purpose
3/30	1500 – 1830	Modifying the support tool
3/31	1500 – 2000	Debugging support tool and updating schedule and risks for website

15.5 Unresolved Problems

- No known unresolved problems.

Chapter 16

Management Report Template Source

Postlab 1
March 31, 2002

16.1 Makefile

```
# $Id: Makefile,v 1.1 2002/02/19 01:25:06 ccf7f Exp $
# Makefile for Flancrest Enterprises' Weekly Management Report

# Files and directories of data
REPORT=mngmnt_report

# Invocations of commands
LATEX=latex
PDFLATEX=pdflatex
LATEX2HTML=latex2html
DVI2TTY=dvi2tty

# Command options
LATEX_OPTIONS=
PDFLATEX_OPTIONS=
L2H_OPTIONS=-split 1 -no_navigation -antialias_text -antialias \
-address "<a href="http://cs-tl3.cs.virginia.edu/cs340-22/">Flancrest \
Enterprises</a> 2001."
D2T_OPTIONS=-w130

default: all clean
```



```
# Run twice to produce toc and refs
dvi::
    $(LATEX) $(LATEX_OPTIONS) $(REPORT).tex
    $(LATEX) $(LATEX_OPTIONS) $(REPORT).tex

# Run twice to produce toc and refs
pdf::
    $(PDFLATEX) $(PDFLATEX_OPTIONS) $(REPORT).tex
    $(PDFLATEX) $(PDFLATEX_OPTIONS) $(REPORT).tex

html::
    $(LATEX2HTML) $(L2H_OPTIONS) $(REPORT).tex

txt::
    make dvi
    $(DVI2TTY) $(D2T_OPTIONS) -o $(REPORT).txt $(REPORT).dvi

#all: dvi html txt pdf # cs-tl3 doesn't have dvitty or latex2htmlx
all: dvi pdf

clean::
    rm -f $(REPORT).log $(REPORT).aux $(REPORT).out texput.log $(REPORT).toc

dist-clean::
    make clean
    rm -rf $(REPORT).dvi $(REPORT).pdf $(REPORT).txt $(REPORT) $(PKGDIR).tar.gz
```

16.2 mngmnt_report.tex

```
% $Id: mngmnt_report.tex,v 1.4 2002/03/31 12:52:49 ccf7f Exp $
\documentclass[tittlepage]{article}

%% Document configuration
\newcommand{\doctitle}{Management Report}
\newcommand{\docdate}{\today}
\newcommand{\doclabnum}{Postlab 0}
\newcommand{\revision}{1} % TODO: use rcs info automatically?

\input{cmds_mngmnt}

\begin{document}

\maketitle
```

```

\input{approval_pledge_toc}

\section{Management Responsibilities}

\begin{mngmntrespon}
Chris & Configuration management and file system control \\ \hline
Emilio & Presentation preparation \\ \hline
Scott & Web development \\ \hline
Thomas & Scheduling and task assignment \\ \hline
Chris and Scott & Document Preparation \\ \hline
\end{tabularx}
\end{mngmntrespon}

\section{Previous Responsibilities}

\begin{prevresp}{Chris}
\input{chris_responsibilities}
\end{tabularx}
\end{prevresp}

\begin{prevresp}{Emilio}
\input{emilio_responsibilities}
\end{tabularx}
\end{prevresp}

\begin{prevresp}{Scott}
\input{scott_responsibilities}
\end{tabularx}
\end{prevresp}

\begin{prevresp}{Thomas}
\input{thomas_responsibilities}
\end{tabularx}
\end{prevresp}

\section{Meeting Reports}
\input{meetings}

\section{Activity Log}

\begin{individualschedule}{Chris}
\input{chris_activity}
\end{tabularx}

```

```

\end{individualschedule}

\begin{individualschedule}{Emilio}
\input{emilio_activity}
\end{tabularx}
\end{individualschedule}

\begin{individualschedule}{Scott}
\input{scott_activity}
\end{tabularx}
\end{individualschedule}

\begin{individualschedule}{Thomas}
\input{thomas_activity}
\end{tabularx}
\end{individualschedule}


\section{Unresolved Problems}
\input{problems}


\end{document}

%% Example code for various things

%\section{Template Examples}
%Some source code written in this document:
%\begin{verbatim}
%#include <iostream>
%using namespace std;

%int main()
%{ return 0; }
%\end{verbatim}

%\subsection{A Subsection}
%\subsubsection{A Subsection of a Subsection}

%Hello.

%\appendix
%\section{Some Stuff}
%I'm in the appendix, hurray.

```

```
%Source code included directly from an external file:
%\includesource{7}{week1/connect_sim.cpp}
```

16.3 preamble.tex

```
\input{cmds_global}

% Setup if statement for pdf work
\newif\ifpdf
\ifx\pdfoutput\undefined
  \pdffalse % not running pdflatex
\else
  \pdfoutput=1 % we are running pdflatex
  \pdftrue
\fi

% Add pdf info
\ifpdf
  \pdfinfo{
    /Title (\doctitle)
    /Author (\groupname)
    /Subject (CS 340)
  }
  \pdfcompresslevel=9
  \pdfcatalog{ /PageMode (/UseNone) } % Don't open a sidepane
\fi

\usepackage{url}
% FIXME: Chris Frost, 1/26: I think this is necessary, but this package
% isn't available on cs-tl3.
%\usepackage{html}

% Enable hyperlinks in document and bookmarks in pdf
% TODO: hyperlinks are annoying, find way to subdue (or get rid of them
% and keep bookmarks)
%\ifpdf
%   \usepackage[bookmarksopen,pdftex]{hyperref}
%\else
%   \usepackage{hyperref}
%\fi

\title{\huge{\textbf{\doctitle}}\}
```

```

\Large{\doclabnum}\\
\large{Revision \revision}}
\author{\LARGE{\textsc{\groupname}}}\ CS~340 Group~22}
\date{\docdate}

\usepackage{geometry}
\geometry{top=1in, bottom=0.3in, left=1.25in, right=1.25in}

\usepackage{tabularx}

\usepackage{fancyhdr}
\pagestyle{fancy}
\rhead{\groupname}
\cfoot{} % The TA's seem to want page numbers on the bottom right, not
        % bottom center.
\rfoot{\thepage}

%\usepackage{rcs}

% For the includesource macro
\usepackage[latin1]{inputenc} % Fixa äö
\usepackage{moreverb}

```

16.4 cmds_global.tex

```

% $Id: cmds_global.tex,v 1.1.1.1 2002/02/03 23:29:36 ccf7f Exp $
% About: General commands for use in all Flancrest Enterprises LaTeX documents.

\newcommand{\groupname}{Flancrest Enterprises}
\newcommand{\ournames}{Chris Frost, Emilio Lahr-Vivaz, Scott Rein, and
Thomas Whanger}

% Use this command, includesource, to include source code from external files.
% (It will include any ascii text actually, not just source code.)
% Use it like: \includesource{tabsize}{filename}. Fwiw, I believe MSVC++
% has a default tabsize of 8.
\newcommand{\includesource}[2]{\verbatiminput[#1]{#2}}

```

16.5 cmds_mngmnt.tex

```

% $Id: cmds_mngmnt.tex,v 1.2 2002/02/19 01:25:06 ccf7f Exp $
% About: Commands for use in all Flancrest Enterprises management reports

```

```

\input{preamble}

% Use to create your weekly activity log, passing your name as an argument.
% Inside of this environment write entries like this:
% 1/27 & 14:00--16:00 & \LaTeX\ macro writing for use in our documents. \\ \hline
% You MUST give a ‘‘\end{tabularx}’’ before the ‘‘\end{individualschedule}’’
\newenvironment{individualschedule}[1]
{
    \begin{center}
    \begin{tabularx}{\textwidth}{|l|l|X|} \hline
    \multicolumn{3}{\textbf{\#1}} \\
    Date & Time & Purpose \\ \hline \hline
    \end{tabularx}
    \end{center}
}

% Keeps count of the number of meetings we’ve had
\newcounter{meetingctr}
\setcounter{meetingctr}{1}

% How to call: \begin{meeting}{date}{time}{location}{attendees}
% Where date is the in form month/day, time in the form
% begin--end (in military notation), and use the macro \ournames
% for attendees if everyone is present.
\newenvironment{meeting}[4]
{ \subsection{Meeting \#\themeetingctr}
  \begin{itemize}
    \item[-] Date: #1
    \item[-] Time: #2
    \item[-] Location: #3
    \item[-] Attendees: #4
  \end{itemize}
  \textbf{Agenda:}
}
{ \addtocounter{meetingctr}{1} }

\newcommand{\results}{\textbf{Results:}}

% Previous postlab responsibilities, accepts name as the parameter
\newenvironment{prevresp}[1]
{
    \begin{center}
    \begin{tabularx}{\textwidth}{|l|X|} \hline

```

```

\multicolumn{2}{|l|}{\textbf{\#1}} \\
Status & Task \\ \hline \hline
}
{ \end{center} }

% Use this environment by inserting data in each row, eg:
% Chris & 1/27 & \LaTeX\ macros for management report \\ \hline
\newenvironment{workassignments}
{
  \begin{center}
  \begin{tabularx}{\textwidth}{|l|l|X|} \hline
  Person(s) & Date Due & Assignment \\ \hline \hline
}
{ \end{center} }

% Used for management responsibilities. Entries are like:
% Chris & Configuration management and file system control \\ \hline
% John & Web \\ \hline
\newenvironment{mngmntrespon}
{
  \begin{center}
  \begin{tabularx}{\textwidth}{|l|X|} \hline
  Person & Responsibility \\ \hline \hline
}
{ \end{center} }

```

16.6 approval_pledge_toc.tex

```

% $Id: approval_pledge_toc.tex,v 1.2 2002/03/05 02:08:57 ccf7f Exp $

\pagenumbering{roman}

\tableofcontents
\pagebreak

\section*{Approval}
We, the members of \groupname, accept this document as representation
of our work for the past week and agree that fair contributions to this
project have been made by all whose signature appear below.\\

\newcommand{\signatures}{
\begin{tabular}{l l}

```

```
\textbf{Name} & \textbf{Signature} \\
& \\
Chris Frost      & \rule{5cm}{0.2pt} \\
& \\
Emilo Lahr-Vivaz & \rule{5cm}{0.2pt} \\
& \\
Scott Rein       & \rule{5cm}{0.2pt} \\
& \\
Thomas Whanger   & \rule{5cm}{0.2pt} \\
\end{tabular}
}
```

```
\signatures
\pagebreak
```

```
\section*{Pledge}
On our honor as students of University of Virginia, we pledge that we
have neither given nor received aid on this assignment.\\
```

```
\noindent \signatures
\pagebreak
```

```
\pagenumbering{arabic}
```

16.7 meetings.tex

```
\begin{meeting}{Month/Day}{1100 -- 1300}{Olsson 001}{\ournames}
\begin{itemize}
  \item This we will cover.
\end{itemize}
\results
\begin{itemize}
  \item This was accomplished.
\end{itemize}
\end{meeting}
```

16.8 chris_activity.tex

```
1/27 & 1230 -- 1430 & \LaTeX\ macro writing for use in our documents.
\\ \hline
```


16.9 emilio_activity.tex

```
1/27 & 1230 -- 1430 & \LaTeX\ macro writing for use in our documents.  
\ \hline
```

16.10 scott_activity.tex

```
1/27 & 1230 -- 1430 & \LaTeX\ macro writing for use in our documents.  
\ \hline
```

16.11 thomas_activity.tex

```
1/27 & 1230 -- 1430 & \LaTeX\ macro writing for use in our documents.  
\ \hline
```

16.12 chris_responsibilities.tex

```
Completed Month/Day & This task was worked on \ \hline
```

16.13 emilio_responsibilities.tex

```
Completed Month/Day & This task was worked on \ \hline
```

16.14 scott_responsibilities.tex

```
Completed Month/Day & This task was worked on \ \hline
```

16.15 thomas_responsibilities.tex

```
Completed Month/Day & This task was worked on \ \hline
```

16.16 problems.tex

```
\begin{itemize}  
  \item No known unresolved problems.  
\end{itemize}
```

Chapter 17

Generic Document Template Source

Postlab 1
February 18, 2002

17.1 generic_doc.tex

```
% $Id: generic_doc.tex,v 1.3 2002/02/19 00:13:19 ccf7f Exp $
% A Generic Flancrest Enterprises Document.
% In this document we include the preamble (which includes global_commands),
% the approval, pledge, and toc sheets, and a document from each person.

\documentclass[titlepage,12pt]{article}

%% Document configuration
\newcommand{\doctitle}{Generic Document}
\newcommand{\docdate}{\today}
\newcommand{\doclabnum}{Postlab 0}
\newcommand{\revision}{1} % TODO: use rcs info automatically?

\input{preamble}

\begin{document}

\maketitle
\input{approval_pledge_toc}

\section{Our Past Summer and Winter Breaks}

\section{Some Source Code}
```

```
\begin{verbatim}
int main()
{
    int a = 42;
    return 0;
}
\end{verbatim}

\end{document}
```

17.2 preamble.tex

```
\input{cmds_global}

% Setup if statement for pdf work
\newif\ifpdf
\ifx\pdfoutput\undefined
    \pdffalse % not running pdflatex
\else
    \pdfoutput=1 % we are running pdflatex
    \pdftrue
\fi

% Add pdf info
\ifpdf
    \pdfinfo{
        /Title (\doctitle)
        /Author (\groupname)
        /Subject (CS 340)
    }
    \pdfcompresslevel=9
    \pdfcatalog{ /PageMode (/UseNone) } % Don't open a sidepane
\fi

\usepackage{url}
% FIXME: Chris Frost, 1/26: I think this is necessary, but this package
% isn't available on cs-tl3.
%\usepackage{html}

% Enable hyperlinks in document and bookmarks in pdf
% TODO: hyperlinks are annoying, find way to subdue (or get rid of them
% and keep bookmarks)
```

```
%\ifpdf
%  \usepackage[bookmarksopen,pdftex]{hyperref}
%\else
%  \usepackage{hyperref}
%\fi

\title{\huge{\textbf{\doctitle}}}\
\Large{\doclabnum}\
\large{Revision \revision}}
\author{\LARGE{\textsc{\groupname}}}\ CS~340 Group~22}
\date{\docdate}

\usepackage{geometry}
\geometry{top=1in, bottom=0.3in, left=1.25in, right=1.25in}

\usepackage{tabularx}

\usepackage{fancyhdr}
\pagestyle{fancy}
\rhead{\groupname}
\cfoot{} % The TA's seem to want page numbers on the bottom right, not
         % bottom center.
\rfoot{\thepage}

%\usepackage{rcs}

% For the includesource macro
\usepackage[latin1]{inputenc} % Fixa ääö
\usepackage{moreverb}
```

17.3 cmds_global.tex

```
% $Id: cmds_global.tex,v 1.1.1.1 2002/02/03 23:27:34 ccf7f Exp $
% About: General commands for use in all Flancrest Enterprises LaTeX documents.

\newcommand{\groupname}{Flancrest Enterprises}
\newcommand{\ournames}{Chris Frost, Emilio Lahr-Vivaz, Scott Rein, and
Thomas Whanger}

% Use this command, includesource, to include source code from external files.
% (It will include any ascii text actually, not just source code.)
% Use it like: \includesource{tabsize}{filename}. Fwiw, I believe MSVC++
```

```
% has a default tabsize of 8.
\newcommand{\includesource}[2]{\verbatiminput[#1]{#2}}
```

17.4 approval_pledge_toc.tex

```
% $Id: approval_pledge_toc.tex,v 1.1.1.1 2002/02/03 23:27:34 ccf7f Exp $
```

```
\pagenumbering{roman}
```

```
\tableofcontents
\pagebreak
```

```
\section*{Approval}
```

We, the members of \groupname, accept this document as representation of our work for the past week and agree that fair contributions to this project have been made by all whose signature appear below.\\

```
\newcommand{\signatures}{
\begin{tabular}{l l}
\textbf{Name} & \textbf{Signature} \\
& \\
Chris Frost & \rule{5cm}{0.2pt} \\
& \\
Emilio Lahr-Vivaz & \rule{5cm}{0.2pt} \\
& \\
Scott Rein & \rule{5cm}{0.2pt} \\
& \\
Thomas Whanger & \rule{5cm}{0.2pt} \\
\end{tabular}
}
```

```
\signatures
\pagebreak
```

```
\section*{Pledge}
```

On our honor as students of University of Virginia, we pledge that we have neither given nor received aid on this assignment.\\

```
\noindent \signatures
\pagebreak
```

```
\pagenumbering{arabic}
```

Chapter 18

Webpage Template Source

Postlab 1
February 3, 2002

18.1 webpage.shtml

```
<!-- $Id: webpage.txt,v 1.1 2002/02/03 17:32:11 ccf7f Exp $ -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head><title>Title</title></head>
<body>

<!-- NOTE: header.php and footer.php are in our base public_html dir,
      you have to indicate where this dir is relative to this file,
      both to find the file and to tell the file where we are (thus the
      dots following the question mark)
      (eg if you are one dir under the base dir: 'virtual="../../header.php?..'')
      (eg if you are two dirs under: 'virtual="../../../header.php?../../..')
-->
<!--#include virtual="header.php?.." -->
<!-- Note: if the section of the website into which this page will go has
      a header of its own as well, be sure to include that where this
      comment is.
-->
</font></font></b></center>

<!-- actual body -->
<hr width="550" size="1" noshade>
</p></p>

<!-- actual content goes here -->
```

```
<!--#include virtual="footer.php?." -->  
</body>  
</html>
```

Chapter 19

Screenshots of FANG

Robot Games
May 1, 2002

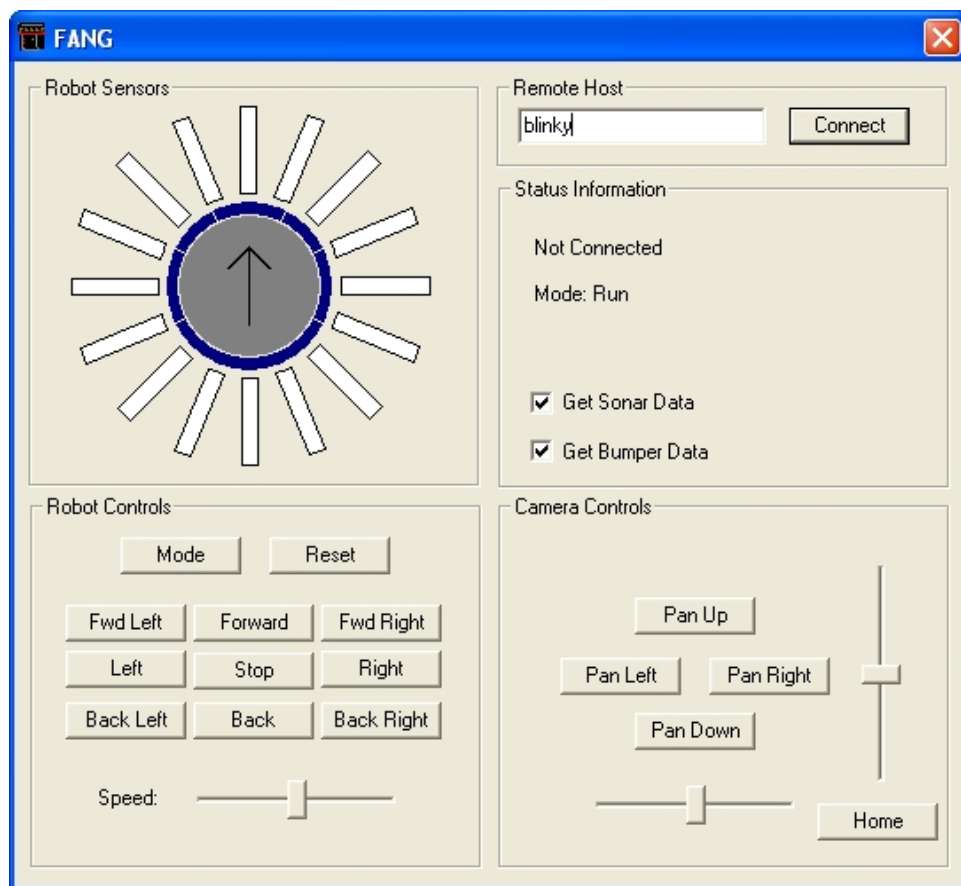


Figure 19.1: The main control window

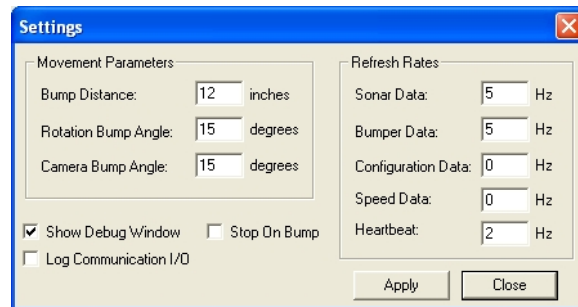


Figure 19.2: The configuration dialog

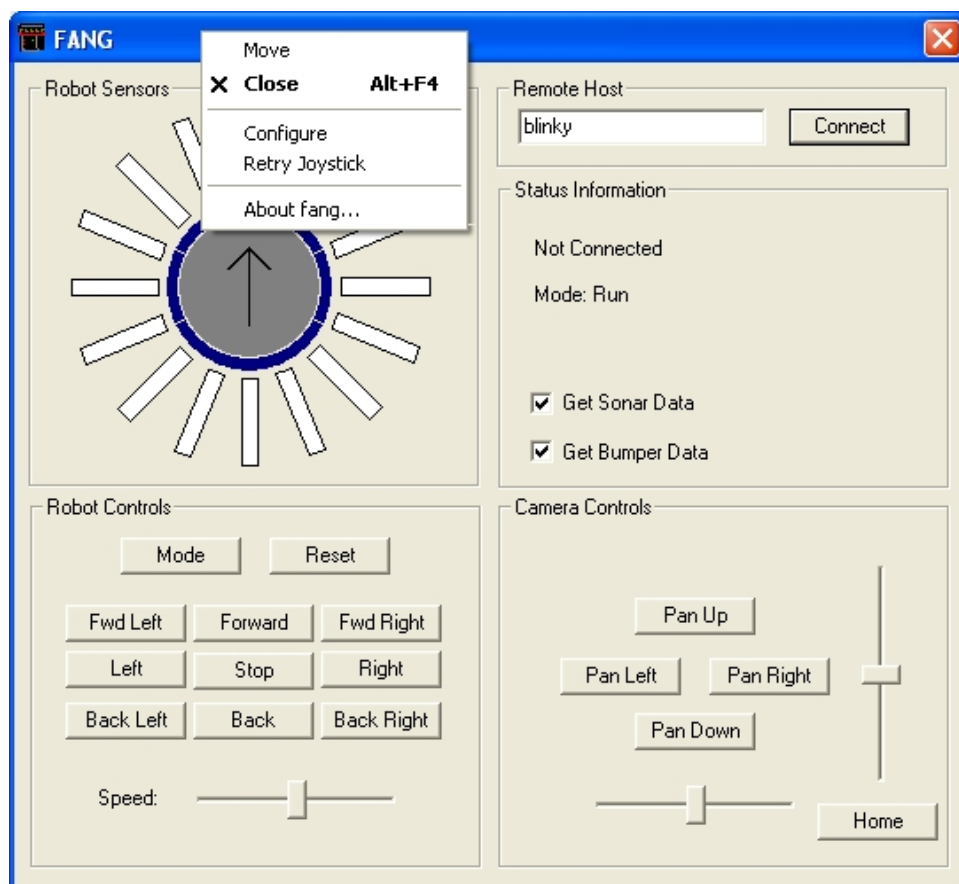


Figure 19.3: Options to open the configuration dialog, retry to acquire a joystick, and to open the about box

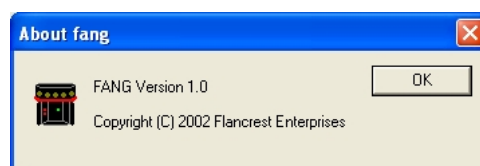


Figure 19.4: The about box



Figure 19.5: The gamepad

Appendix A

Selected Website Pages

Appendix B

FANG's Source

Appendix C

FURL's Source

Appendix D

Partner Group's Client Interface Specification