```
# $Id: makefile,v 1.4 2002/04/01 19:41:50 ccf7f Exp $
CPP=g++
CFLAGS  = -Wall -pedantic -g
LFLAGS =
LIBS =
OBJECTS = command.o server.o
BINARY = server

all: $(BINARY)

clean:
    rm -f $(BINARY) $(OBJECTS)

command.o: command.cc command.h
    $(CPP) $(CFLAGS) command.cc -c -o command.o

server.o: server.cc
    $(CPP) $(CFLAGS) server.cc -c -o server.o

server: $(OBJECTS)
    $(CPP) $(CFLAGS) $(OBJECTS) -o server $(LFAGS) $(LIBS)
```

```
/*
** server.cc -- a stream socket server support tool
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/param.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
//#include <fstream.h>
#include <vector>
#include <string>
using namespace std;

#include "command.h"

const int MAX_ARGS   = 30;      // maximum number of args to a command

const int MaxBuffer  = 1024;
const int ServerPort = 6942;

string parseMessage( string msg )
{   char name[30];   // Q: is 30 the same as MAX_ARGS?
    vector<string> argv;   // need to dynamically size argv
    {
        string empty;
        for(int i=0; i<MAX_ARGS; i++)
            argv.push_back(empty);
    }
    int i = 0, j = 1, k = 0; // j iterates through entire msg
    Command *cmd;
    string actReturn;

    // initialize array
    for (i = 0; i < 20; i++) { // Q: from where does 20 come?
        argv[i] = "             "; // Q: from where does 30 come?
    }
    i = 0;

    if (msg[0] != '(' || msg[msg.size()-1] != ')') {
        printf("ERROR: Check message format!\n");
        return "";
    }

    // require that the command be all lower or uppercase char's
    // read the command name
    while ( (65 <= msg[j] && msg[j] <= 90) ||
            (97 <= msg[j] && msg[j] <= 122) ) {
//      printf("code: %d, i: %d, j: %d\n", (int)msg[j], i, j);
        name[i++] = msg[j++];
    }
    name[i] = '\0';  // null terminate the string

    i = 0;
    // read the arguments
    while (1) {
        if (msg[j] == ' ') {
            j++;
            k++;
            i = 0;
            continue;
        } else if (msg[j] == ')') {
            break;
        } else {
```



```
            (argv[k-1])[i] = msg[j];
            j++;
            i++;
        }
    }
    cmd = cmdMaker( string(name), k, argv );
    if (cmd) {  // parse did not fail
        cout << "Received command: " << cmd->getName() << endl;
        actReturn = cmd->Act();
        delete cmd;
        return actReturn;
    }
    return "";
}

int main ( int argc, char *argv[] )
{   char    message[MaxBuffer + 1];
    int Socket, AcceptedSocket, len, startFlag;
    char    singleChar[2];
    int   i;

    struct sockaddr_in  server;
    struct hostent      *HostInfo;
    //
    // Open a socket and start listening for connections
    //
    memset( &server, 0, sizeof( struct sockaddr_in ) );
    gethostname( message, MaxBuffer );
    HostInfo = gethostbyname( message );
    if ( HostInfo == NULL )
    {
        perror( "could not get hostname" );
        exit( 1 );
    }
    server.sin_family = HostInfo->h_addrtype;
    server.sin_port = htons( ServerPort );
    Socket = socket( AF_INET, SOCK_STREAM, 0 );
    if ( Socket < 0 ) {
        perror( "could not create socket" );
        exit( 1 );
    }
    if ( bind( Socket, ( struct sockaddr * ) &server, sizeof( struct sockaddr_in ) )
< 0 ) { perror( "could not bind socket" );
        exit( 1 );
    }
    listen( Socket, 0 );

    // Handle requests one at a time
    while ( 1 ) {
        AcceptedSocket = accept( Socket, NULL, NULL );
        if ( AcceptedSocket < 0 )
            continue;
        printf("Made connection\n");

        // code to read messages, acting as a FSM
        memset( message, 0, MaxBuffer );  // zero the buffer
```

```
        singleChar[0] = 0;
        startFlag = 0;  // determine if reading in a command
        i = 0;
        while ( ( len = read( AcceptedSocket, singleChar, 1 ) ) > 0 ) {
            // ignore these characters
            if (singleChar[0] == 13 || singleChar[0] == 10)
                continue;

            if ( singleChar[0] == '(' && startFlag == 0 ) {
                i = 0;
                message[i] = '(';
                startFlag = 1;
            }
            else if ( singleChar[0] == ')' && startFlag == 1 ) {
                message[++i] = ')';
                message[++i] = '\0';
                string reply = ParseMessage(message);
                send( AcceptedSocket, reply.c_str(), reply.size(), 0 );
                memset( message, 0, MaxBuffer );
                i = 0;
                startFlag = 0;
            }
            else if ( startFlag == 1 ) {
                    message[++i] = singleChar[0];
            }
            else {
                printf( "ERROR: Malformed command!\n" );
            }
        }
        close( AcceptedSocket );
    }

    close( Socket );    // This should never be reached
    return 0;
}
```

```
#ifndef COMMAND_H
#define COMMAND_H

#include <vector>
#include <string>
using namespace std;
class Command {
    public:
        Command();
        virtual ~Command();
        //      Command(string name, int argc, string argv[]);
        virtual string Act() = 0;
        string getName();
        int getArgc();
        vector<string> getArgv();
        string getArg(int n);
    protected:
        string m_name;
        int m_argc;
        vector<string> m_argv;
};
// returns an Ack to a subclass of Command
string ackMaker(string cmdName, vector<string> cmdArgs);
// returns a Command* to a subclass of Command
Command* cmdMaker(string name, int argc, vector<string> argv);

class cmd_da : public Command
{
   public:
        cmd_da(string name, int argc, vector<string> argv);
       virtual string Act();
};
class cmd_vm : public Command
{   public:
        cmd_vm(string name, int argc, vector<string> argv);
       virtual string Act();
};
class cmd_pr : public Command
{   public:
        cmd_pr(string name, int argc, vector<string> argv);
       virtual string Act();
};
class cmd_rotateRobot : public Command
{
   public:
        cmd_rotateRobot(string name, int argc, vector<string> argv);
       virtual string Act();
};
class cmd_sp : public Command
{
   public:
        cmd_sp(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_st : public Command
{
   public:
        cmd_st(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_moveCamera : public Command
{
```



```
   public:
        cmd_moveCamera(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_bumpCamera : public Command
{
   public:
        cmd_bumpCamera(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_getBp : public Command
{
   public:
        cmd_getBp(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_getRc : public Command
{
   public:
        cmd_getRc(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_getSp : public Command
{
   public:
        cmd_getSp(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_getSn : public Command
{
   public:
        cmd_getSn(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_getState : public Command
{
   public:
        cmd_getState(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_heartBeat : public Command
{   public:
        cmd_heartBeat(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_init : public Command
{   public:
        cmd_init(string name, int argc, vector<string> argv);
        virtual string Act();
};
class cmd_disconnect : public Command
{   public:
        cmd_disconnect(string name, int argc, vector<string> argv);
        virtual string Act();
};
#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include "command.h"
#include <string.h>
#include <vector>
#include <string>
using namespace std;

string ackMaker(string cmdName, vector<string> argv) {
    string ack = string("ack ") + cmdName + string("\n");
    return ack;
}
Command* cmdMaker(string name, int argc, vector<string> argv) {
    Command *cmdNew = NULL;

    if (name == "vm") {
        cmdNew = new cmd_vm(name, argc, argv);
        return cmdNew;
    }
    else if (name == "pr") {
        cmdNew = new cmd_pr(name, argc, argv);
        return cmdNew;
    }
    else if(name == "rotateRobot") {
        cmdNew = new cmd_rotateRobot(name, argc, argv);
        return cmdNew;
    }
    else if(name == "da") {
        cmdNew = new cmd_da(name, argc, argv);
        return cmdNew;
    }
    else if(name == "sp") {
        cmdNew = new cmd_sp(name, argc, argv);
        return cmdNew;
    }
    else if(name == "st") {
        cmdNew = new cmd_st(name, argc, argv);
        return cmdNew;
    }
    else if (name == "heartBeat") {
        cmdNew = new cmd_heartBeat(name, argc, argv);
        return cmdNew;
    }
    else if(name == "disconnect") {
        cmdNew = new cmd_disconnect(name, argc, argv);
        return cmdNew;
    }
    else if(name == "init") {
        cmdNew = new cmd_init(name, argc, argv);
        return cmdNew;
    }
    else if(name == "moveCamera") {
        cmdNew = new cmd_moveCamera(name, argc, argv);
        return cmdNew;
    }
    else if(name == "bumpCamera") {
        cmdNew = new cmd_bumpCamera(name, argc, argv);
        return cmdNew;
    }
    else if(name == "getBp") {
        cmdNew = new cmd_getBp(name, argc, argv);
        return cmdNew;
    }
    else if(name == "getRc") {
        cmdNew = new cmd_getRc(name, argc, argv);
        return cmdNew;
    }
```



```
    else if(name == "getSp") {
        cmdNew = new cmd_getSp(name, argc, argv);
        return cmdNew;
    }
    else if(name == "getSn") {
        cmdNew = new cmd_getSn(name, argc, argv);
        return cmdNew;
    }
    else if(name == "getState") {
        cmdNew = new cmd_getState(name, argc, argv);
        return cmdNew;
    }
    else {
        printf("ERROR: Unknown command!\n");
        return NULL;
    }
}
Command::Command() {  // constructor
}
Command::~Command() {
}
string Command::getName() {
    return m_name;
}
int Command::getArgc() {
    return m_argc;
}
vector<string> Command::getArgv() {
    return m_argv;
}
string Command::getArg(int n) {
    return m_argv[n-1];  // note that argument numbering starts at 1
}
cmd_da::cmd_da(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_da::Act() {
    if(m_argc != 1){
        printf("Error: wrong number of args to da");
        return "(invalidArgs da)";
    }
    if(!( (atoi(getArg(1).c_str()) <= 3600) && (atoi(getArg(1).c_str()) >= 0) )) {
        printf("Error: Steering Angle out of Range");
        return "(invalidArgs da)";
    }
    return ackMaker(m_name, m_argv);
}
cmd_vm::cmd_vm(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_vm::Act() {
   if(m_argc != 2) {
        printf("Error: wrong number of args to vm");
```

```
        return "";
    }
    /*
    if(!( (getArg(1) <= "400") && (getArg(1) >= "-400") ))
        printf("Error: right wheel velocity out of range");
    if(!( (getArg(2) <= "400") && (getArg(2) >= "-400") ))
        printf("Error: right wheel velocity out of range");
    */
    return ackMaker(m_name, m_argv);
}
cmd_pr::cmd_pr(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
        m_argv = argv;
}
string cmd_pr::Act() {
    if(m_argc != 2){
        printf("Error: wrong number of args to pr");
        return "(invalidArgs pr)";
    }

    if(!( (atoi(getArg(1).c_str()) <= 32000) && (atoi(getArg(1).c_str()) >= -32000) )
)
        printf("Error: right wheel step out of Range");

    if(!( (atoi(getArg(2).c_str()) <= 32000) && (atoi(getArg(2).c_str()) > -32000)))
        printf("Error: left wheel step out of Range");
    return ackMaker(m_name, m_argv);
}
cmd_rotateRobot::cmd_rotateRobot(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_rotateRobot::Act() {
    if(m_argc != 2){
        printf("Error: wrong number of args to rotateRobot");
        return "";
    }
    if(!( (atoi(getArg(1).c_str()) <= 3600) && (atoi(getArg(1).c_str()) >= -3600)))
        printf("Error: rotate angle is out of range");
    return ackMaker(m_name, m_argv);
}
cmd_sp::cmd_sp(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_sp::Act() {
    if(m_argc != 2){
        printf("Error: wrong number of args to sp");
        return "";
    }
    if(!( (atoi(getArg(1).c_str()) <= 400) && (atoi(getArg(1).c_str()) >= 0) ))
        printf("Error: right wheel speed out of range");
    if(!( (atoi(getArg(2).c_str()) <= 400) && (atoi(getArg(2).c_str()) >= 0) ))
        printf("Error: left wheel speed out of range");
    return ackMaker(m_name, m_argv);
```



```
}
cmd_st::cmd_st(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_st::Act() {
    if(m_argc != 2){
        printf("Error: wrong number of args to st");
        return "";
    }
    if(!( (atoi(getArg(1).c_str()) == 0) || (atoi(getArg(1).c_str()) == 1) ))
        printf("Error: Robot Stop out of range");
    if(!( (atoi(getArg(2).c_str()) <= 1) || (atoi(getArg(2).c_str()) >= 0) ))
        printf("Error: Camera Stop out of range");
    return ackMaker(m_name, m_argv);
}
cmd_moveCamera::cmd_moveCamera(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_moveCamera::Act() {
    if(m_argc != 2){
        printf("Error: wrong number of args to moveCamera");
        return "";
    }
    if(!( (atoi(getArg(1).c_str()) <= 450) && (atoi(getArg(1).c_str()) >= -450) ))
        printf("Error: pan speed out of range");
    if(!( (atoi(getArg(2).c_str()) <= 450) && (atoi(getArg(2).c_str()) >= -450) ))
        printf("Error: tilt speed out of range");
    return ackMaker(m_name, m_argv);
}
cmd_bumpCamera::cmd_bumpCamera(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_bumpCamera::Act() {
    if(m_argc != 2){
        printf("Error: wrong number of args to bumpCamera");
        return "";
    }
    if(!( (atoi(getArg(1).c_str()) <= 32000) && (atoi(getArg(1).c_str()) >= -32000) )
)       printf("Error: pan distance is out of range");
    if(!( (atoi(getArg(2).c_str()) <= 32000) && (atoi(getArg(2).c_str()) >= -32000) )
)       printf("Error: tilt distance is out of range");

    if((! (atoi(getArg(3).c_str()) <= 450) && (atoi(getArg(3).c_str()) >= 0) ))
        printf("Error: pan rotational speed is out of range");
    if(!( (atoi(getArg(4).c_str()) <= 450) && (atoi(getArg(4).c_str()) >= 0) ))
        printf("Error: tilt rotational speed is out of range");
    return ackMaker(m_name, m_argv);
}
```

```
cmd_getBp::cmd_getBp(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_getBp::Act() {
    if(m_argc != 1) {
        printf("Error: wrong number of args to getBp");
        return "";
    }

// TODO: error checking on Arg 1

    return ackMaker(m_name, m_argv);
}
cmd_getRc::cmd_getRc(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_getRc::Act() {
    if(m_argc != 1) {
        printf("Error: wrong number of args to getRc");
        return "";
    }

// TODO: error checking on Arg 1
    return ackMaker(m_name, m_argv);
}
cmd_getSp::cmd_getSp(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_getSp::Act() {
    if(m_argc != 1) {
        printf("Error: wrong number of args to getSp");
        return "";
    }

// TODO: error checking on Arg 1
    return ackMaker(m_name, m_argv);
}
cmd_getSn::cmd_getSn(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_getSn::Act() {
    if(m_argc != 1) {
        printf("Error: wrong number of args to getSn");
        return "";
    }

// TODO: error checking on Arg 1
    return ackMaker(m_name, m_argv);
}
cmd_getState::cmd_getState(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
```



```
string cmd_getState::Act() {
    if(m_argc != 1) {
        printf("Error: wrong number of args to getState");
        return "";
    }

// TODO: error checking on Arg 1
    return ackMaker(m_name, m_argv);
}
cmd_heartBeat::cmd_heartBeat(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_heartBeat::Act() {
    if (m_argc != 1) {
        printf("Error: wrong number of args to heartBeat");
        return "";
    }
    return "(heartBeat)";
}
cmd_init::cmd_init(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_init::Act() {
    if(m_argc != 6) {
        printf("Error: wrong number of args to pr");
        return "";
    }

    if(!( (atoi(getArg(1).c_str()) <= 400) && (atoi(getArg(1).c_str()) >= 0) ))
        printf("Error: right wheel speed out of Range");

    if(!( (atoi(getArg(2).c_str()) <= 400) && (atoi(getArg(2).c_str()) >= 0) ))
        printf("Error: left wheel speed out of Range");
    if(!( (atoi(getArg(3).c_str()) <= 400) && (atoi(getArg(3).c_str()) >= 0) ))
        printf("Error: Pan Camera Speed out of Range");

    if(!( (atoi(getArg(4).c_str()) <= 400) && (atoi(getArg(4).c_str()) >= 0) ))
        printf("Error: Tilt Camera Speed out of Range");
    if(!( (atoi(getArg(5).c_str()) <= 1) && (atoi(getArg(5).c_str()) >= 0) ))
        printf("Error: Stop on Bump out of Range");

// TODO: error checking on Arg 6
    return ackMaker(m_name, m_argv);
}
cmd_disconnect::cmd_disconnect(string name, int argc, vector<string> argv) {
    m_name = name;
    m_argc = argc;
    m_argv = argv;
}
string cmd_disconnect::Act() {
    if(m_argc != 0) {
        printf("Error:wrong number of arguments");
       return "" ;
   }return ackMaker(m_name, m_argv);
}
```

May 02, 02 0:16 **Table of Content** Page 1/1

Table of Contents