

# Telerobotics over the Internet

Mark Fienup and Holly Jongdyk

## Introduction

The purpose of this grant was to develop the essential software infrastructure to support telerobotics over the Internet. We have successfully completed the design, development, and implementation of a prototype telerobotics system. The telerobotics prototype consists of a Rug Warrior Pro™ mobile robot which carries a small wireless video camera and a Compaq iPAQ with a wireless ethernet card. A remote operator is able to direct the robot's motion via the Internet while monitoring the robot's progress using "live" images from the wireless camera.

This grant has helped enable the Computer Science Department at UNI to begin building a new research capability in the area of telerobotics with the development of reusable software components for the telerobotics communication and the computer-human interface.

The long-term goal of this research is to investigate the feasibility of using telerobotics techniques to implement a Personal Home Assistant (PHA). The PHA as envisioned would be able to perform simple household chores under direct control of a remote operator. A list of possible tasks would include:

- assist individuals with physical handicaps
- adjusting the heating/cooling thermostat
- closing/opening windows
- checking on correct operation of sump pump during thunderstorms
- checking on the security of a house
- feeding and watering the household pet (e.g., dog/cat/goldfish)
- letting out/in the dog
- programming the VCR
- starting microwave, oven, etc. to heat pre-prepared foods

To accomplish these tasks, a much more sophisticated robot with the ability to precisely manipulate objects is needed. Never the less, this project is a small step in providing the infrastructure to support such a project.

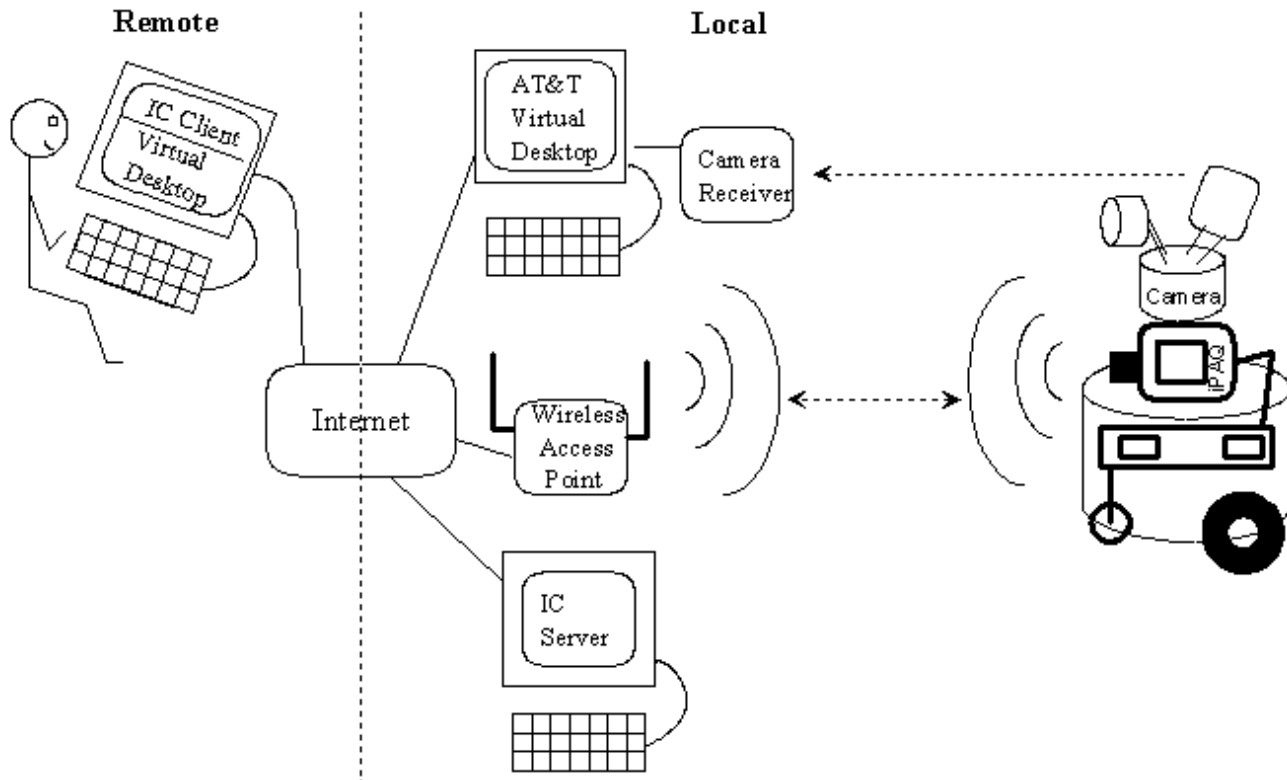
## Architectural Overview

To program behaviors for the RugWarrior Pro robot, we are using interactive C (IC). IC is a compilation environment for Motorola 6811 based systems. It implements a subset of ANSI C language and was developed by Randy Sargent and Fred Martin for the MIT LEGO Robot Design Competition. Interactive C compiles the RugWarrior Pro robot behavior into byte codes to download to the p-code interpreter running in the microcontroller of the robot. The Rug Warrior Pro robot contains a 6811 microcontroller (MC68HC11) with a built-in serial port for communication with a host computer. Since we did not want the robot to be limited in motion by a physical serial connection, it was equipped with an iPAQ Pocket PC with a wireless ethernet card. The iPAQ's serial interface was then connected to the robot's serial port as shown in Figure 1.

An UNI student assembled the Rug Warrior robot from a kit for this research. A commercial version of interactive C for Windows was included with the kit. From this IC version, we are using the p-code interpreter and the library files of IC functions. These functions provide interfacing capabilities with the hardware on the robot controller board.

In the commercial version of IC, users can only issue commands or type in expressions from the keyboard of

the host PC while the robot is connected via a serial cable a desktop PC. We needed IC to be able to receive user inputs from a remote machine connected the host PC. Thus we needed to modify the input and output routines in the source code of IC. We obtained the IC source code of a freeware UNIX/Linux version from the MIT Media Laboratory ftp server. Next, we describe the modifications made to IC.



**Figure 1. Overview of Architecture**

## The TCP/IP socket programming

A socket is a programming interface that enables a communication channel between unrelated processes running on different machines over a network. Once a socket is properly initialized, it can be used to read (write) bytes of data from (to) another socket. The winsock, windows sockets, library implements the socket functionality for WinCE (Pocket PC) on the iPAQ, whereas it is a part of the standard C library in most Unix systems (glibc for Linux). There are different styles of socket that one can create, but we used connection-oriented sockets (style `SOCK_STREAM`) exclusively.

In general, a server creates a socket and waits listening for any incoming connections. The client must 'connect' to this socket in order to establish a communication. The server must 'accept' the connection request before any data transmission can occur. The server's original socket does not become part of the connection. In fact 'accept' creates a new socket, which participates in the communication with this particular client. The server's original socket remains available for listening for further incoming connection. The new socket can be used to transfer bytes in either direction using the generic 'read' and 'write' system calls or the socket specific 'recv' and 'send' functions.

We have modified the freeware version 2.8 of IC to include a server to listen for connections from the remote workstation and a client to connect with the iPAQ server.

## Modifications to IC source code

Before modification, IC first compiles user's interactive c codes into byte codes and stores them in a temporary buffer. The function used is `compile (...)`. The function `board_download (...)` is used to download these byte codes to the microcontroller on the robot. These byte codes were loaded through a serial line to the the robot. The IC code had to be modified to a wirelessly transmit these byte codes to a server program running on the iPAQ which then downloads them through a serial line to the robot.

Originally, `Board_download (...)` uses `io_flush (...)` in the `iob.c` file and the `io_get_char_multiple_inputs_internal_n (...)` functions in `iob_unix.c` to write to and read from, respectively, the serial interface. `Io_read` and `io_write` were modified taking into account the protocol between IC client and the iPAQ server.

In order to download, write, n bytes to robot, we send five bytes plus n bytes of data to the iPAQ server. The first byte contains the instruction 'w', the next four bytes contain the number of bytes to download (i.e. n), and the remaining bytes are the actual byte codes. Then we read the response from the iPAQ server. The five response bytes should contain 'w' and the actual number of bytes the iPAQ server was able to download to the microcontroller. In case of error, the iPAQ server returns 'e' in the first byte.

```
Io_flush
  if (there is data for robot) {
    get the size of data;
    send 'w'+size of data(4 bytes)+the data to iPAQ server;
    read and analyze the server response;
  }
```

Appendix A contains the `io_flush(...)` function.

When we need a byte from robot, `io_getchar_multiple_inputs_n_internal(...)` function first checks if there is a byte stored in the internal buffer from a previous read operation. If the buffer is empty, it then asks the iPAQ server to read from the serial port ('r'+amount to read+timeout value to use during serial read). The amount of data read by the iPAQ server is stored in the internal buffer. Both of these functions uses `io_read()` and `io_write()` functions which reads and writes raw bytes to the connection socket, respectively.

## iPAQ server program

A server program for iPAQ was written to enable IC to remotely communicate with the robot's microcontroller. The server was written in embedded C, which is a compiler for embedded systems running WinCE. The iPAQ uses the Microsoft ActiveSync technology for serial communication with the robot. The server communicates with a single client at a time maintaining a persistent connection until the client disconnects. The iPAQ server and its clients will implement the following protocol:

### Protocol implemented by iPAQ server:

write: (to iPAQ server)

'w' + no of bytes iPAQ server will write to serial port (4 bytes) + the bytes

'r' + no of bytes iPAQ server will read from the serial port (4 bytes) + port read timeout in millisecond (4 bytes)

read: (from iPAQ server)

'w' + no of bytes written to serial port (4 bytes)

'r' + no of bytes read from the serial port (4 bytes) + the bytes

'e' + xxxx (4 bytes) in case of error while reading from or writing to the serial port.

The server program first reads only five bytes sent by IC to find out the nature of the request and the size of the remaining data to read from the client. It then uses the standard Windows serial communication routines to satisfy the request. Finally, the result is sent back to the client.

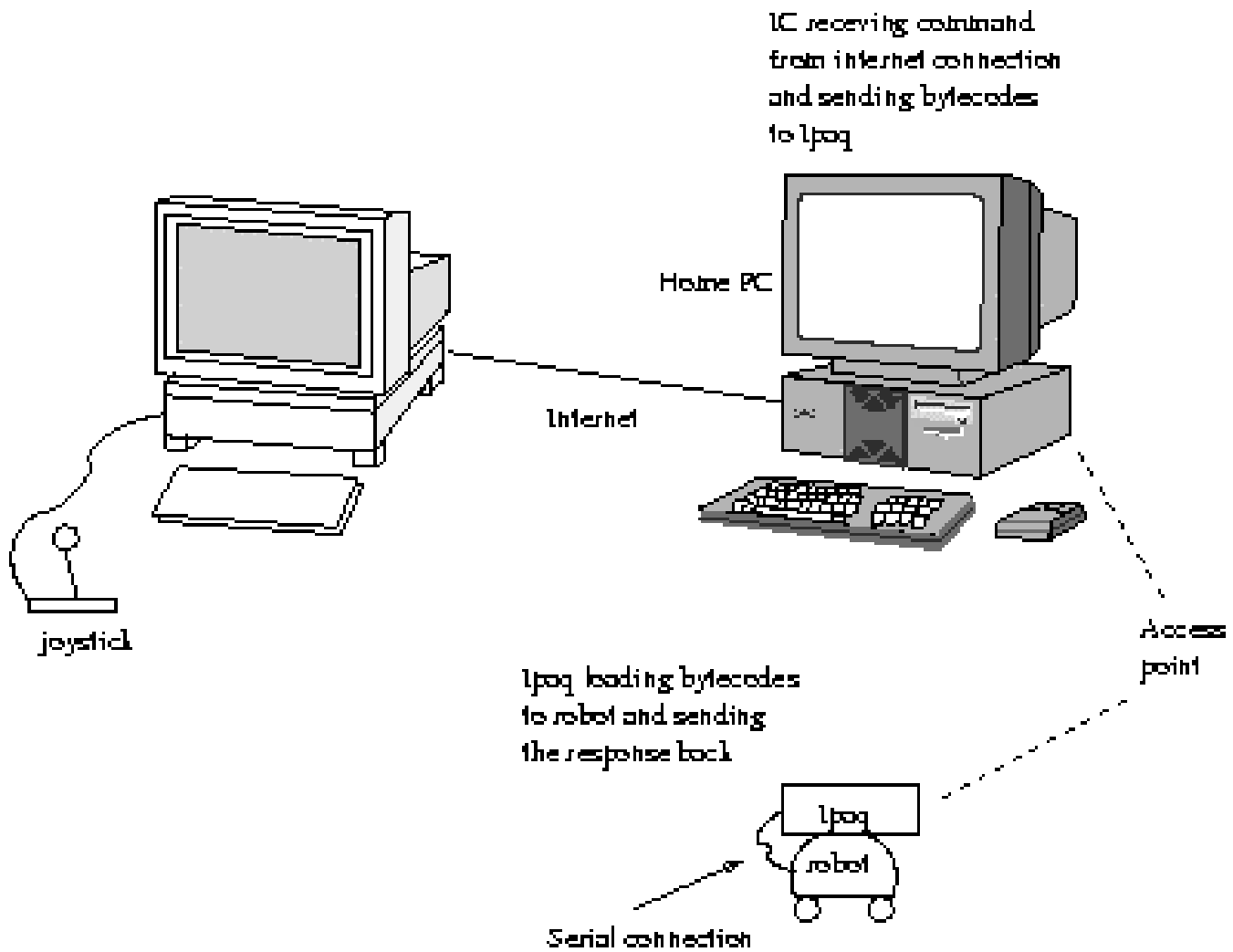


Figure 2. Details of Communication Path

## Remote Client to Communicate to IC

The remote client connects to the IC server socket and send interactive C commands to the robot and receives its response back. Currently the remote host will telnet to the local host. A GUI application and/or a joystick may be used to control the robot instead of telnet.

The following is a list of IC commands that the remote operator can issue to control the robot:

start_move( );	Starts the initial process on the robot and the robot's motors
forward ( );	Move robot forward
backward( );	Move robot backward
right( );	Start the robot turning toward the right
left( );	Start the robot turning toward the left
halt( );	Stops the robot from moving or turning
get_bumpr ( );	Get robot's bumper state
t_velocity = 100;	Set translational velocity of robot's motor
r_velocity = 50;	Set rotational velocity of robot's motor
stop_move ( );	Stop robot's process
state;	Return the state of the robot

## Image Server on Local Workstation

A wide-angle camera was purchased from X10 to ride on the robot and transmit wirelessly a video signal to a video receiver. The video receiver is connected to the usb port of the local workstation. The X10 software captures and displays images on the local workstation. We are currently using Virtual network computing (VNC) from AT&T to transport these images to the remote workstation.

## Robot behavior control in IC

The behavior function continuously monitors user's input and takes appropriate action whenever necessary to synchronize its state with that desired by the user. The user has to explicitly start this process using a built-in command of IC. The user's input is represented by three persistent global variables: command, t\_velocity and r\_velocity.

Persistent variables keep their state in between runs of your software. The corresponding variables used to keep track of the current state are state, t\_vel, and r\_vel. Command holds the value the user wishes to send to the motor controller. The possible values for this variable are encoded as follows:

Command = 1 => go forward  
Command = 2 => go backward  
Command = 3 => start turning to the left  
Command = 4 => start turning to the right  
Command = 0 => stop current motion until a new command is received

Any other value will terminate the behavior control process. The 't\_velocity' and 'r\_velocity' are the desired translational and rotational velocities, respectively. The robot moves forward or backward at the velocity

specified by 't\_velocity'. An additional persistent variable 'bmpr' is used to store the current state of the bumper switches which can then be read by the user. Once the robot starts, it will continue to move in the direction of the command given until the command changes or the robot hits an object. The states of the bumper switches are read at each iteration of the while loop, if the robot is in motion. The motors of the robot are stopped once the bumper(s) collide with an object. Appendix C contains the pseudocode for the robot behavior.

## Conclusions and Future Work

The purpose of this grant was to develop the essential software infrastructure to support telerobotics over the Internet. During the project we were successfully able to design, develop, and implement a prototype telerobotics system. The telerobotics prototype consists of a Rug Warrior Pro™ mobile robot which carries a small wireless video camera and a Compaq iPAQ with a wireless ethernet card. A remote operator is able to direct the robot's motion via the Internet while monitoring the robot's progress using "live" images from the wireless camera.

The long-term goal of implementing a more general Personal Home Assistant (PHA) able to perform simple household chores under direct control of a remote operator will require a more sophisticated robot. This robot must be able to precisely manipulate objects as needed. This is a direction of future work.

This grant has helped enable the Computer Science Department at UNI to begin building a new research capability in the area of telerobotics with the development of reusable software components for the telerobotics communication and the computer-human interface.

## References

1. AT & T Virtual Network Computing website, <http://www.uk.research.att.com/vnc/>
2. Bonner, *Network Programming with Windows Sockets*, (1996), Prentice Hall PTR, Upper Saddle River, NJ.
3. Jone, Flynn, and Seiger, *Mobile Robots: Inspiration to Implementation*, 2<sup>nd</sup> edition, (1998), A. K. Peters, Ltd., Natick, MA.
4. Jones, *Rug Warrior Pro Assembly Guide*, A. K. Peters, Ltd., Natick, MA.
5. MIT Media Laboratory ftp site to download UNIX/Linux version of Interactive C, <ftp://chelona.media.mit.edu/pub/projects/interactive-c/unix/>
6. Nelson, *Serial Communications Developer's Guide*, (2000), IDG Books Worldwide, Inc., Foster City, CA.

## APPENDIX A

### Modified IC Code. Note `io_read` is a new function.

```
/* s - stream created by io_open_iPAQ() in ic.c */
/* buf - buffer containing data to write */
/* len - length of data in buf */

Int io_write(IOStream *s, char *buf, long len)
{
    long wrote=0;
    long rc;
    int max;
    struct timeval tv;
    fd_set wr_set, rd_set;

//Loop until everything written out

    while(wrote<len){
        FD_ZERO(&rd_set);
        FD_ZERO(&wr_set);

        FD_SET(s->stream, &wr_set);
        max=s->stream;

        tv.tv_sec = 2;
        tv.tv_usec = 0;
        if(-1 == select(max+1, &rd_set, &wr_set, NULL, &tv)) {
            fprintf(stderr, "Error:select in io_write\n");
            return wrote;
        }

        if(FD_ISSET(s->stream, &wr_set)){
restart:
            rc=write(s->stream , buf+wrote, len-wrote);
            if (rc < 0) {
                if(errno==EAGAIN) continue;
                if(errno==EINTR) goto restart;
                fprintf(stderr, "Error:write in io_write\n");
                exit(1);
            } else if(rc==0){
                fprintf(stderr, "iPAQ server closed connection\n");
                exit(1);
            }
            wrote+=rc;
        }
    }
    return wrote;
}
```

```

/*reads data from the iPAQ_server */
Int io_read(IOStream *s, char *buf, long len)
{
    Int nread=0;
    int rc;
    int max;
    struct timeval tv;
    fd_set wr_set, rd_set;

    while(nread<len){
        FD_ZERO(&rd_set);
        FD_ZERO(&wr_set);

        FD_SET(s->stream, &rd_set);
        max=s->stream;

        tv.tv_sec = 2;
        tv.tv_usec = 0;
        if(-1 == select(max+1, &rd_set, &wr_set, NULL, &tv)) {
            fprintf(stderr, "Error:select in io_read\n");
            buf[nread]=0;
            return nread;
        }

        if(FD_ISSET(s->stream, &rd_set)){
restart:
            rc = read(s->stream, buf+nread, len-nread);
            if(rc < 0) {
                if(errno == EAGAIN) continue;
                if(errno == EINTR) goto restart;
                fprintf(stderr, "Error:read in io_read\n");
                exit(1);
            } else if(rc==0) {
                fprintf(stderr, "iPAQ server closed connection\n");
                exit(1);
            }
            nread +=rc ;
        }
    }
    buf[nread]=0;

    return nread;
}

```

```

void io_flush(IOStream *s)
{
    char buffer[1029];

```

```

int ndata=0, i=0;
if (!BYTEQUEUE_EMPTY(&s->outbuf)) { //we have something for robot
    ndata= BYTEQUEUE_LENGTH(&s->outbuf); //size of data

    //prepend 'w'+ the size of data to the actual data
    sprintf(buffer, "%c%4d", 'w', ndata);
    bytequeue_remove(&s->outbuf, &buffer[5], ndata);

    i=ndata+5;
    if(i==io_write(s, buffer, i)){ //send the bytes to iPAQ server
        if(5==io_read(s, buffer, 5)){ //read iPAQ server's response
            //check server's response
            if(buffer[0]=='w' && atoi(&buffer[1])!=ndata)
                fprintf(stderr, "iPAQ unable to write every byte\n");
            else if(buffer[0]=='e')
                fprintf(stderr, "Error downloading to robot\n");
        }
    }
}
}
}

```

```

Int io_getchar_multiple_inputs_n_internal(IOStream *input, Int timeout)
{
    char buffer[1024];
    Int nread;
    int size;

    if (!BYTEQUEUE_EMPTY(&input->inbuf)) {
        return bytequeue_remove_char(&input->inbuf);
    }

    /*---- tell iPAQ server to read the serial port-----*/
    size=BYTEQUEUE_SPACE(&input->inbuf);
    sprintf(buffer, "%c%4d%4d", 'r', size, (int) timeout);
    if(9 != io_write(input, buffer, 9)){
        fprintf(stderr, "Unable to write 9 bytes\n");
        return EOF;
    }

    /*read iPAQ_server response 'r'+ no of bytes read from serial line*/

    if(5==io_read(input, buffer, 5)){
        if(buffer[0]=='r'){
            size = atoi(&buffer[1]); //4 bytes contain the size
            if(size>0)
                nread = io_read(input, buffer, size);//read the data
            else
                return EOF;
        }
    }
}

```

```
    if(nread>0){//got some data from robot, add it to buffer
        bytequeue_add(&input->inbuf, buffer, nread);
        return bytequeue_remove_char(&input->inbuf);
    } else{
        return EOF;
    }
} else if(buffer[0]=='e'){
    fprintf(stderr, "Error io_read(): iPAQ returned error\n");
    return EOF;
}
}

return EOF;//select or read failed
}
```

## Appendix B

### Original IC code (major portion that was changed.)

```
Int io_write(IOStream *s, char *buf, long len)
{
    long wrote;
    long pos= 0;
    /* Loop until all written out */
    while (pos < len) {
        wrote= write (s->stream ? s->stream : 1, buf+pos, len-pos);
        if (wrote < 0) {
            fprintf(stderr, "Error in io_write\n");
            return -1;
        }
        pos += wrote;
    }
    return len;
}

void io_flush(IOStream *s)
{
    char buffer[1024];
    Int i= 0, wrote;
    if (!BYTEQUEUE_EMPTY(&s->outbuf)) {
        i= BYTEQUEUE_LENGTH(&s->outbuf);
        bytequeue_remove(&s->outbuf, buffer, i);
        /*if (s->stream == 0) return;*/
        wrote= io_write (s, buffer, i);
        if (wrote != i) {
            fprintf(stderr, "Error on write in iob_unix: stream %s. Tried to write %ld chars, ret= %ld\n",
                s->name,
                i, wrote);
        }
    }
}

Int io_getchar_multiple_inputs_n_internal(IOStream **inputs, Int *which, Int timeout, Int n)
{
    Int nfound;
    Int i, max= 0;
    char buffer[1024];
    fd_set read_set, write_set, exceptional_set;
    struct timeval timeval_timeout;

    for (i= 0; i < n; i++) {
        if (!BYTEQUEUE_EMPTY(&inputs[i]->inbuf)) {
            *which= i;
        }
    }
}
```

```

        return bytequeue_remove_char(&inputs[i]->inbuf);
    }
}

timeout *= io_delay_factor;

timeval_timeout.tv_sec= (long) (timeout / 1000);
timeval_timeout.tv_usec= 1000L * (long) (timeout % 1000);

FD_ZERO(&read_set);
FD_ZERO(&write_set);
FD_ZERO(&exceptional_set);

for (i= 0; i< n; i++) {
    FD_SET(inputs[i]->stream, &read_set);
    if (inputs[i]->stream > max) max= inputs[i]->stream;
}

nfound= select(max+1, &read_set, &write_set, &exceptional_set,
               timeout < 0 ? 0 : &timeval_timeout);

if (nfound <= 0) {
    *which= EOF;
    return EOF;
}

for (i= 0; i< n; i++) {
    if (FD_ISSET(inputs[i]->stream, &read_set)) {
        Int nread;
        *which= i;
        nread= read(inputs[i]->stream, buffer, BYTEQUEUE_SPACE(&inputs[i]->inbuf));
        if (nread <= 0) {
            /* *which= EOF; */
            return EOF;
        }
        else {
            /*if (inputs[i]->stream != 0) {
                write(1, buffer, nread);
            }*/

            bytequeue_add(&inputs[i]->inbuf, buffer, nread);
        }
        return bytequeue_remove_char(&inputs[i]->inbuf);
    }
}

printf("Didn't find the readable file\n");
exit(1);
/* does not reach here */
return 0;
}

```

## Appendix C

### Pseudocode of Robot Behavior

```
while( ! terminate ) {
  if ( current translational velocity != desired translational velocity )
  {
    update current translational velocity;
    if ( robot in forward or backward motion )
      make the new velocity effective
  }
  if ( current rotational velocity != desired rotational velocity )
  {
    update current rotational velocity;
    if ( robot in left or right turning motion )
      make the new velocity effective;
  }
  if ( current state != stopped || desired state != stopped )
  {
    read state of bumper switches;
    if ( current state != stopped && there is a collision )
    {
      stop motion;
      update current and desired states to stop;
    }
  }
  if ( current state != desired state )
  {
    if (desired state == stop )
    {
      stop motion;
      update current state;
    }
    else if ( desired state == move forward && left and right switches not closed )
    {
      move forward at current translational velocity;
      update current state;
    }
    else if ( desired state == move backward && back switch not closed )
    {
      move backward at current translational velocity;
      update current state;
    }
    else if ( desired state == left turn && left switch not closed )
    {
      start turning left at current rotational velocity;
      update current state;
    }
  }
}
```

```
    }  
    else if ( desired state == right turn && right switch not closed )  
    {  
        start turning right at current rotational velocity;  
        update current state;  
    }  
}  
} // end while
```