UMEÅ UNIVERSITY
Department of Computing Science
Thomas Hellström

Handling the Khepera robots

The robots are very expensive and very delicate. For the equipment to survive, it is necessary that everyone is familiar with and obey to the following instructions and rules.

Booking

The robots are booked in a schedule attached on the wall outside the Computing Science corridor on the fourth floor.

Check-out and return

The robots are kept at studentexpeditionen over nights. The first group for the day checks out their robot there and returns it either to the same place or to the next group in turn. The last group for the day returns the robot to studentexpeditionen or to some employee at the computing science department who takes care of returning it to studentexpeditionen.

General

- Never lift the robot such that it will fall to the floor if you drop it. Always hold it over the table or your maze. If the robot falls to the floor it will be severely damaged. However, these injuries are minor in comparison to the ones that will be inflicted on you when the incident is discovered.
- NEVER lift the robot in the cable ! The cables are also very delicate and you do not appreciate working with malfunctioning cables!
- When disconnecting: Pull the connectors instead of pulling the cables!
- The camera module may not be detached from the Khepera since the connectors are very delicate and easy to ruin.
- The communication speed is set to 38400 baud on the Khepera switches and should not be changed.

Connection

The robot is connected to the PC as follows:

1. The long thin cable should be connected to the rotating device. The other end of the cable is connected to the gray interface.
2. The cable that goes from the rotating device is connected to the Khepera robot. Please observe the latch on the red connector that fits into a small hole on the Khepera connector!
3. The long cable with a 9 pin male connector is connected to the serial port marked COM1 or COM2 on the PC (the choice affects the value of the *port_id* parameter in the comm_open routine)
4. The other side of the cable is connected to the interface.

5. The power adapter is connected to the mains 230 volt.
6. The cable from the power adapter is plugged into the interface

The rotating device sometimes fails, You may in such case remove it and connect the thin cable straight from the Khepera to the interface.

Disconnection

The important thing when disconnecting the cables is that the power adapter is first disconnected (either from the interface or from the wall) This operation can also be used as a “master reset” of the robot.

General

- The robot and all its equipment should to be in the black case when it is being moved around.
- The robot must never be left unattended. You are responsible for it even if you are not present!
- Never ever Fika over the robot. It does not like cookie crumbs and a drip of coffee may be lethal!
- The switch underneath the robot should be in OFF position (internal batteries disconnected).
- If you are nice to the Khepera, it will also be nice to you. It always does its best to understand the commands you are sending to it, and misunderstandings are almost always due to HUMAN ERRORS.

UMEÅ UNIVERSITY
Department of Computing Science
Thomas Hellström

Drivers for Serial Communication

Introduction

Drivers are sub routines or functions that are used to communicate with hardware. A common standard is serial communication that sends data in serial bit-streams between units. One example is the communication between a PC and an external modem. In the case with our Khepera robots, the communication is used to control the motors of the robot and to read values from the sensors and the camera. Besides routines for transferring data in both directions, a number of other routines are necessary for opening/closing the communication channels, for checking the number of characters in the data buffers etc.
Often an ASCII protocol is used. All transmitted messages are in this case streams of (mainly) human readable characters and a few control characters such as line-feed and carriage return. Other protocols are binary and use data coded at bit-level. The Khepera uses an ASCII protocol.
The physical connection between Khepera and PC is a cable from the serial port of the PC to (the gray colored) communication module that comes with the Khepera. This module connects further to the Khepera through a multi colored cable. **NOTE: The power adapter must NOT be connected to the communication module when the other cables are being connected or disconnected.**

Layers

Drivers perform data communication between the computer and the connected equipment, and are often implemented as modules handling different layers of abstraction in the communication.

Low-level:
Communicates with the hardware, i.e.: reading and writing data with no concern of the meaning of the data. Routines to initiate and close the communication are also on this level.

Mid-level:
Calls low-level routines, extracts data (read routines) and packs data (write routines). E.g.: `sendfloat(v)` may be a routine sending a real number (according to a hardware dependent standard) to the hardware.

High-level:
Calls routines from the mid level and performs commands close to the application. E.g.: `turn(degrees)` could be a routine executing communication necessary to make the robot turn. To be regarded as a driver routine, the function has to perform a common and hardware related task.

Important notes:

Speed:

The driver routines are normally called very often in an application program. In principle, execution speed should therefore be of high priority. However, in many cases, such as in the Khepera case, the communication intensity is still very low, and speed is not really an issue. Other applications may require drivers to be written in C or even assembler to achieve necessary speed.

Stability:

Drivers may never cause a stop in the program.
Drivers may not assume that data always arrives or that is has the specified format.
Faulty or missing data should be handled by resending commands or requests.
Refer to the section Retransmission below.

Error handling:

Write routines: Should check input parameters to avoid data outside the specified borders to be sent to the hardware.
Read routines: Received data has to be checked at least at the low level: number of parameters and type of parameters (alphanumeric or numeric).
General: Check sums are used for certain commands (not with the Khepera though). Every transmitted data entity (command, record etc) is followed by a checksum with a format that can be checked by the receiver. In cases where a checksum is not included in the communication, a format check can often serve the same purpose (albeit not as safe).
Example: A request for data from the IR sensors on the Khepera robot is sent as the following string from the computer: `"N<lf>"`. I.e.: the character "N" followed by the line feed character (ascii 10). A short moment later, the robot will reply with a string containing values for the 8 IR sensors. E.g.: `"n,123,43,23,334,677,876,54,32<lf>"`. There is no checksum, but the receiving program can make sure the string starts with the "n" character and ends with a line feed. When values are extracted from the string, the number of integers can be checked (should be 8 in this case).
Retransmission: If the transmission did not work (no or erroneous reply), data should be sent again. After a fixed number of such attempts, a time-out should be issued. This error condition should be transferred upwards in the call hierarchy as a status or error variable.

Timing aspects

Since external equipment is connected, a certain degree of indeterminism is, from the computer's perspective, introduced. After requesting data from an external unit (e.g. an instrument or a robot), the reply time may differ depending on the status and load of the external equipment. In addition to this delay, the communication time has to be considered. If the communication speed is 9600 baud, a maximum of 870 characters can be transferred per second (8 data bits, 2 stop bits, 1 start bit). If large amounts of data have to be communicated, the delay has to be estimated and affect the timeout behavior previously described. The sender and receiver may also get out of phase such that a received answer belongs to an old request. The remedy to this is extended error handling. However, in our robot applications, a missed, incorrect or delayed command does most often not affect the function of the program since new commands with the same or almost the same commands are being transmitted many times per second.

Support for program development

Debug printouts at many levels can simplify both development and usage of the driver routines. A global variable can be used to control debug printouts without messing up the parameter lists of the routines.

Drivers for the Khepera robot

Read about the communication protocol for the Khepera and the camera module K213 in their respective manuals in the compendium.

Command format

All commands to the Khepera have a common format:

```
PC                          KHEPERA
<CMD>,<arg><lf>      ->
                     <-   <cmd>,<arg-return><lf>

<CMD> is an upper-case letter defining the type of command
<cmd> is the same letter but in lower case
<arg> is a comma-separated list with arguments to the command
The comma sign should not be sent if <arg> is empty
<arg-return> is a comma-separated list with results from the Khepera
The comma sign is not returned if <arg-return> is empty
<lf> is the ascii character "line feed" ( char(10) in Matlab ).
```

Example (set the robot engine's speed):

```
PC                          KHEPERA
D,5,15<lf>           ->
                     <-     d<lf>
```

Commands to K213 and other "turrets":

A separate command "T,x" is tells the Khepera that the next command should be interpreted by a turret with turret_id = x. The camera K213 has turret_id 2.

Available subroutines

For the Khepera drivers we will use a set of routines that can open the communication channel, send a command and wait for a reply from the Khepera. The manufacturer of the Khepera (K-team) supplies these routines. They implement the low-level and some of the mid-level functionality above, and you will, as part of one assignment, write the other necessary routines. The drivers from K-team are downloadable from the course web page.

`port = kopen([ com_port, baud_rate, timeout])`

Opens the communication channel for serial communication through COM1 or COM2 on the PC.

```
Parameters:
com_port : 0 for COM1, 1 for COM2.
baud_rate: 9600 for 9600 baud etc.
time_out : Controls the  timeout function  for ksend
           Number of seconds before ksend "times out".
```

Example for COM2 with 38400 baud and 0.1 seconds timeout:

```
» port = kopen([1,38400,0.1])
```

`answer = ksend(s,port,1)`

Sends a command `s` to the Khepera and waits for a reply that is returned as a string.
If no answer arrives before the set timeout (set in `kopen`), `ksend` returns an empty string. The parameter `port` is the handle returned by `kopen`.

```
E.g:
>> answer = ksend(['N' 10],port,1)
n,123,32,123,43,2,33,323,666
>>
```

The example sends the command "Read proximity sensors" to the Khepera that returns a string containing data from the 8 IR sensors. Note the linefeed character that has to end all commands.

`kclose(port)`

Closes the communication channel opened by `kopen`. The parameter `port` is the handle returned by `kopen`.

UMEÅ UNIVERSITY
Department of Computing Science
Thomas Hellström

Assignment 1

This assignment should be performed in groups consisting of 1-3 students. The necessary low-level routines kopen, ksend and kclose are downloadable from the course web page.

Part a:

a1: Write the following drivers in Matlab for communication with the Khepera robot. More information can be found in the document ”Drivers for serial communication”.

comm_open

```
function status = comm_open(port_id, baud, timeout, debugg)
%     function comm_open(port_id, baud, timeout, debugg)
%
%     port_id - The communication port to be used. 0 for COM1, 1 for COM2.
%     baud    - Set the transfer speed. E.g.: 9600.
%     timeout - No. Of seconds before the read routines timeout.
%     debugg  - Level of debugg printouts. 0 for no printouts.
```

The first three parameters are the same as for kopen. debugg can set a global variable that the other routines can use for debug printouts. comm_open calls kopen and defines a global variable port that the other routines can use.
comm_open can start by calling comm_close to close a possibly opened line. The Khepera robots are set to the highest communication speed **38400 baud**.

comm_close

```
function status = comm_close
% Close the communication port.
```

Uses the global variable port.
Note: If kclose is called with a non-initialised port variable, the computer may have to be rebooted. Test this with ~isempty(port) before calling kclose.

KhepCom

```
Function [ok,values] = KhepCom(cmd,arg, N, turretID)
% Sends a command cmd with arguments arg to the Khepera robot.
% Retries at most 5 times or until N numbers are read correctly from the
% Khepera. The read values are returned in values.
% A vector with N NaN is returned if unsuccessful.
% ok is returned == 1 iff successful
% Example 1:  ok = KhepCom('D',[12 15],0) % Set speed
% Example 2: [ok, values] = KhepCom('O',[],8) % Read ambient light sensors
% Example 3: [ok, values] = KhepCom('N',[],64,2);% Read K213
%
% the turretID parameter is used to send commands to turrets instead of to
% the main Khepera. Refer to the T command for details
```

a2:

When `KhepCom` is written and tested, the following routines should be easy to write: (about 1 line each):

```
function ok = set_speed(speed)
%     Set motor speed. The unit is the pulse, corresponding to 8 mm/s
%
%     speed - 1x2 vector containing: the speed for the left motor,
%                                    the speed for the right motor
%     See the function D in the Khepera User Manual for details


function [values, ok] = read_prox
%      Read the 10 bit values from the proximity meter sensors.
%      A vector with 8 NaN are returned if unsuccessful
%      See the function N in the Khepera User Manual for details



% function [values, ok] = read_amb
%      Read the 10 bit values from the ambient light meter sensors.
%      A vector with 8 NaN are returned if unsuccessful
%      See the function O in the Khepera User Manual for details

% function [values, ok] = read_vision
%      Read the 64 8 bit values from the K213 camera turret.
%      A vector with 64 NaN are returned if unsuccessful
%      See the function N in the Khepera User Manual for details
```

Finally: Write a test program using the developed routines. To display the camera image, imshow can be used:

```
[image, ok] = read_vision;
figure(2)
xmul=30; ymul=6;
imshow(repmat(reshape(repmat(image',ymul,1),64*ymul,1),1,xmul),[0 255]);
```

The call to repmat duplicates image along y and x to a suitable size.

General tips:

- `sscanf(str, ',%d')` can be used to extract integers from `str`.
- Use try och catch for error handling in Matlab.
- Test KhepCom properly with simulated correct and incorrect readouts from the Khepera. Then test with the robot connected.
- Use the debugger! Make a breakpoint where ksend is called to check the format of the transmitted string. Step forward and check the string that ksend returns.
- Remember: The routines have to work no matter what the Khepera returns (including no return). Correct time-out handling is very important.

Part b:

Implement a "Breitenberg vehicle" in Matlab for the Khepera.
Since there are 8 infra-red reflex sensors, a somewhat more complicated mapping function from stimuli to response has to be used. Use a simple perceptron with linear activation function for each engine:

$$m_L = \sum_{i=1}^{8} w_i \cdot r_i + w_0$$

$$m_R = \sum_{i=1}^{8} v_i \cdot r_i + v_0$$

w_0 och v_0 are offsets that will make the robot move even if the reflex sensors return zero (which means there are no close obstacles). You may add an amplification factor (about 1/400) for all weights to make it easier to experiment with the weights.
The program should use the routines comm_close, comm_open, read_prox and set_speed from part a.

Experiment with different weights to get the following behaviors:
1: ”Avoid obstacle”: the robot should turn away from obstacles
2: ”Follow the wall” : the robot should be able to find its way out of a maze by following the left (or right) wall.

Can the symmetry left/right be used to simplify the weight settings?
Can the weights be calibrated by showing the robot a wall or an obstacle?
Can you find other ways to simplify the weight setting?

More information about Breitenberg vehicles:
http://www.mindspring.com/~gerken/vehicles

Report:

- Normal written CS report with source code in appendix.
- Part b is inspected by another group (that your group don’t inspect).
- The signed inspection protocol is then handed in with the report.
- Remember to write your name and email on the front page of the report!

Inspection protocol

Assignment 1 - Course Intelligent Robotics

Group members in block letters

…… ……

Tests:

1.
Verify assignment part b:
1: ”Avoid obstacles”
2: ”Follow the wall”

2.
Disconnect the cable from the power unit.
The robot should now stop (if not: Contact me and national TV)
Now return the cable.
The robot should now resume normal operation
If the cable is disconnected for too long, the program should time-out and stop.

We have conducted the above tests to our full satisfaction

…… ……

Names in block letters:

…… ……

……
Date

Umeå universitet 901 87 Umeå. Telefon: 090-786 7759 . E-post: thomash@cs.umu.se