

## Homework 5 Appendix: Communication Between Robots

Note: You are *not* required to use these utilities. Or, if you want to change them, you may. They are just provided for your convenience, in case you want to use them.

---

On the course website are 2 files provided for you: `communicate.h` and `commsExample.cc`:

<http://www.cs.utk.edu/~parker/Courses/CS594-spring07/Assignments/communicate.h>

<http://www.cs.utk.edu/~parker/Courses/CS594-spring07/Assignments/commsExample.cc>

These files give you the basic routines needed to communicate between multiple processes, such as robots in Player/Stage. These routines are set up to send UDP datagrams between two robot processes. (The reason why this uses UDP rather than TCP is that, in general, UDP offers better performance than TCP on real robots, which can break. We can have problems with hung processes on real robots if, for example, one robot fails during the application. We won't go into the details here.)

The `commsExample.cc` file shows you how to use these communications messages. The gist is this: You enter (as command line parameters) the ID and PORT number of the current robot's process, as well as the ID and PORT number of the 2<sup>nd</sup> ("friend") robot's process. Then, the code sets up the socket for communication. When you want the current robot to send a message to the other robot, your code must format that message (using the "send\_cmd" process). When you want the current robot to receive a message from the other robot, your code must use the "recv\_cmd" process to read and decode the message.

### About message formats

The provided code *does not* provide the complete message format that you'll need for this homework. Instead, it shows you how you can create messages according to your own format. The basic message format is as follows:

TS\$R!

where:

T: 1 character message type (currently 'F' means target found, and 'C' means application is complete (stop the program))  
S: ID of sender (converted to characters)  
\$: special delimiter symbol separating fields  
R: ID of recipient (converted to characters)  
!: special delimiter symbol denoting end of message

So, for example, the actual message sent/received might be:

F1\$2!

However, for this homework, you'll need to beef up this message format to add in the x,y position of the target position. So, ultimately, your sent/received messages would look something like:

```
TS$R$xxx.xx$yy.y!
```

A specific message might be:

```
F1$2$10.5$-1.4!
```

where `xxx.xx` is the `x` position of the target and `yy.y` is the `y` position of the target. It is your job to add this additional information to your messages. You may want to make use of functions such as `atof` for this purpose. You'll need to make these changes in 2 places: in `send_cmd` to format the message to be sent, and in `recv_cmd` to decode the received message. You can add in any message types as you like and find useful.

Note that in processing the messages, the code is set up to only accept messages from the "friend" robot, and only if the message has the current robot's ID in the recipient field. This is a good safeguard, to ensure that the robot only responds to valid messages intended for itself.

### **Command line parameters**

To run this code, you'll need to enter in the current robot's ID and port number, and the "friend" robot's ID and port number as command line arguments. So, for one process, you'll have something like:

```
linux> ./commsExample 1 6000 2 6001
```

and for the second process, you'll have something like:

```
linux> ./commsExample 2 6001 1 6000
```

These command lines will allow these two processes to send and receive messages from each other.