SMR-CL.

Small Mobile Robot Control Language, SMR-CL is a simple interpreted language intended for control of mobile robots. The language supports sequences of basic robot actions with multiple stop criterion as well as standard mathematical expressions. The system has two kinds of variables system variables that reflects the state of the vehicle and user variables that are created the first time they are used in an assignment clause. The language is intended to be used in the tactical layer just over the control layer i.e. it is fast enough to shift between control strategies and react to state changes in real time. The hope is that it will be able to bridge the gab between the hard real-time demands of the control layer and the soft realtime demands of the planning (strategic) layer. The language may be used in two ways either as scripts run directly from text files or as a command language through a socket interface. Most commands are available in both modes but a few are only available in one of the modes.

Basic format.

The language is line oriented i.e. it is interpreted line by line. The basic format is

command parameters [@v velocity] [@a acceleration] [: controlconditions]

where [ ] means optional.

command	is the desired command
parameters	is a number of parameters required by the command. The parameters can be numbers, variables or string litterals (characters within " ")
velocity	desired vehicle velocity in m/s (variable or number)
acceleration	desired acceleration in m/s^2 (variable or number)
controlconditions	(expression) [| (expression)]* a list of ORed stopconditions, the interpreter will execute the next command if one or more of the conditions are true. The system variable $*condition* will hold the number of the first true condition (from left)

Examples:

fwd 2.0 @v0.5 @a0.05

move forward 2 m with acceleration 0.05 m/s^2 max velocity 0.5 m/s

drive @v0.5 :($drivendist > 4)| ($cmdtime > 12)

drive at velocity 0.5 m/s until 4 m is driven or time used is 12 seconds

turn 90

turn 90 degress around center

Motion control commands

The following motion control commands exist:
all commands will use current velocity and acceleration references as max values

fwd d	move forward d meters
turn b ["rad"]	turn b degrees (radians if "rad")around center (between the driven wheels)
turnr r b ["rad"]	turn b degrees(radians if "rad") with turning radius r m
drive [x y th] ["rad"]	drive with current speed along the line defined by point (x,y) and direction th, if the “rad” parameter is given the angle is measured in radians else in degrees. The distance from the current position to a line through x,y perpendicular to th is given in $targetdist. If x,y and th are omitted the current position and direction of the vehicle is used.
stop	stop vehicle and keep position and direction
idle	stop vehicle and keep speed at zero.
followline "type"	follow line using the method described by type: bl left side of black line bm middle of black line br right side of black line wl, wm, wr as above with white line.
followwall "side" dist	follow a wall to the side of the vehicle indicated by side: l left , r right dist distance to wall in meters
resetmotors	resets the motor modules
ignoreobstacles	make the vehicle ignore obstacles in the next command
targethere	Sets the targetpose for fwd and turn commands to current pose.

Examples

% square program1 (comments are indicated with the % character)
fwd 1 @v0.3
turn 90
fwd 1
turn 90
fwd 1
turn 90
fwd 1
turn 90

The vehicle runs in a square with side length 1m and max velocity 0.3 m/s

Flow control.

The following flow control statements are implemented:

The label, goto and if statements are only available in scripting mode.

label "labelname"

goto "labelname" continues execution at the line with label "labelname"

call "labelname" continues execution at the line with label "labelname" and returns to the line after the call line when executing a **return** command

return return to the line after the latest **call** command.

if (exp) "labelname" jump to "labelname" if expression is true (not zero)
switch (exp)
statements these statements are executed if exp < 0.5
case 1
statements these statements are executed if 0.5 < exp < 1.5
case 2
statements these statements are executed if 1.5 < exp < 2.5

case n
statements these statements are executed if n-0.5< exp< n+0.5
endswitch

The switch statement must not be nested.

% the rest of the line after % is a comment

example:

```
% square program2
n=4
label "start"
fwd 1 @v0.3
turn 90
n=n-1
if (n >0)   "start"
```

Expressions and Variables.

Variables are automaticly created when used as left side of an assignment clause:

a=1.5

creates a variable with name a and assigns the value 1.5 to it. Array variable must be declared using the syntax:

array “name” length

The cammand

array “myarray” 4

will create an array named myarray with four elements that are accessed with index 0 to 3

a[2]=3

will assign 3 to the third element of the array.

The language supports normal arithmetic expressions:

+ - * /	
> >= < <= == !=	
|	logical or
&	logical and
( )	
=	assignment

Functions.

sin(x)
cos(x)
tan(x)
atan(x)
atan2(y,x)
ln(x)
exp(x)
sqrt(x)
abs(x)
normalizeanglerad(x) normalizes x to the range]-pi;pi]
normalizeangledeg(x) normalizes x to the range]-180;180]

Examples:
x=1.0
y=3
a=2.5*sin(x)+sqrt(y)
array “ar1” 4
a[0]=0
a[3]=3

Logging of Data.

The language supports simultaneus logging of up to 9 variables. Logging is started with the log command and the result is stored in the text file 'log' in the current directory when the run is stopped. The variables are sampled every 0.01 seconds and stored as a line for each sample time i.e. they may be loaded into e.g Matlab or Excel.

log "varname" ["varname"]*

Example:
log "$odox" "odoy" "$odoth"

Miscellaneous.

eval exp [; exp]* print the results of the expressions on the screen in scriptmode and returns the values in socket mode.

wait time makes the program wait for **time** seconds

trans x0 y0 th0 x y th

Transforms coordinates between coordinate system A and B. (x0,y0,th0) is the origin and orientation of A measured in B-coordinates. (x,y,th) are A-coordinates of a pose and the resulting B-coordinates are stored in the system variables $res0, $res1 and $res2. The transformation is given by:

$$\begin{pmatrix} \$res0 \\ \$res1 \end{pmatrix} = \begin{pmatrix} \cos(th0) & -\sin(th0) \\ \sin(th0) & \cos(th0) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x0 \\ y0 \end{pmatrix}$$

$$\$res2 = th + th0$$

vision “command text”

sends the “command text” to the cameraserver if started. The server might return data in the default variables $vis0 to $vis9
If the command text is $string the string generated by the latest stringcat command will be used

get “guidemark” returns the data of a guidemark seen by the camera. The cameraserver must be started.

laser “command text”

sends the “command text” to the laserserver if started. The server might return data in the default variables $l0 to $l9
If the command text is $string the string generated by the latest stringcat command will be used.

stringcat parameters

concatenates the parameters to a string saved in $string. If a parameter is a string it is added unchanged if the parameter is a number or a variable it is converted to a string and added. The command is intended to be used with the “laser” or “vision” commands

example: stringcat “send “ 9 “ apples”
will generate the string “send 9 apples”

Socket interface.

The socket interface is also lineorientated. The interface consists of the following special commands:

getevent [time-out] fetches the next event in the event queue. If the queue is empty the command will return after 'time-out' seconds with the message 'eventtimeout'. If no time-out is given the command will return immediately.

putevent "text " generates an event with the text 'userevent text' where text is the text entered between the '"s

flushcmds flushes all commands from the command queue

logoff logs off and leaves the robot ready for a new logon

exit logs off and stop the robot controlprogram

addwatch “name” type :(expression)

puts a watch in the watch table. If **expression** becomes true an event 'watch **name** fired **time**' , where name is the name given in addwatch and time is the time where the expression became true.

When a command is sent to the socket the robot will respond with 'IDnn queued' where nn is a unique number. Execution of the command will generate a number of events that may be retrieved with the command 'getevent' . If the command is finished in one sample the event Idnn assignment will be generated. If the command takes more samples to execute the start of execution will be signalled with 'IDnn started' and end of execution with 'IDnn stopcond xx' where xx is the value of the stopcondition.

It is also possible to download a named smrcl-script which then can be invoked using the name. The following commands are used:

beginplan	name	name is the name wanted for the script
plan	smrcl line	downloads the script line by line
endplan		ends the script
deleteplan	name	deleted the script with the given name
runplan	name	starts execution of the script

The downloaded scripts can use all available smr-cl commands including flow control.
In the current version up to 10 plans can be downloaded at the same time.

System variables.

Odometry:

$odox	x-coordinate of vehicle position (m)
$odoy	ycoordinate of vehicle position (m)
$odoth	direction of vehicle (rad)
$odovelocity	velocity of vehicle (based on odometry measurements)
$ododist	distance driven since start of vehicle (m)
$drivendist	distance driven during the current command (m)

IR-sensors:

all distances are measured from the sensor itself.

$irdistleft	distance (m) measured by the left side sensor
$irdistfrontleft	distance (m) measured by the front left sensor
$irdistfrontmiddle	distance (m) measured by the front middle sensor
$irdistfrontright	distance (m) measured by the front right sensor
$irdistright	distance (m) measured by the right side sensor

$irl	raw sensor value measured by the left side sensor
$irfl	raw sensor value measured by the front left sensor
$irfm	raw sensor value measured by the front middle sensor
$irfr	raw sensor value measured by the front right sensor
$irr	raw sensor value measured by the right side sensor

$cmdtime	time spent in the current command (sec)
$batteryvoltage	batteryvoltage (v) lowpass filtered with a time constant of a second
$supplyvoltage	voltage of the external power supply (v) lowpass filtered with a time constant of 0.1 second
$time	system time in seconds (Linux time of day)
$motionsstatus	status telling if the robot is stopped by an obstacle
$motorstatusr	motor status for the right motor
$motorstatusl	motor status for the left motor

Linesensor:

$line0 to $line7	calibrated linesensor values
$lineraw0 to $lineraw7	raw linesensor values
$linepos	position of found line
$blacklinefound	true if a black line is detected (used to find a line approched from the side)
$crossingblackline	true if a black line is crossing the line the vehicle follows

Guidemark:

$guidemarkok	contains the return code from a **get "guidemark"** command
$gmkx	x-coordinate of the guidemark
$gmky	y-coordinate of the guidemark
$gmkxz	z-coordinate of the guidemark
$gmkomega	guidemark angle

$gmkophi	guidemark angle
$gmkkappa	guidemark angle

Robot Control Program.

The program implementing SMRCL is called MRC**.** It is started by logging on to the robot using ssh and starting the program from the terminalwindow.

MRC [options] [scriptfile]

The program has the following options:

-c	calibration. This will start a calibration menu where the linesensor, the irsensor and the odometry may be calibrated. The calibration results will be saved in files in the directory calib which must exist before calibration.
-t n	start the program as a socket server on port 31000+n. (n is an integer)
-s n	start the program with a simulated smr
-v off	runs the program in silent mode (no printing on the screen).