KTH Machine Design

Autonomous Parking Project

Frank Wang, Yavor Trasiev

Content

Abstract

Our task was to develop and implement a system with processor, bus and sensors capable of controlling the RCV so that it can start in front of the garage door, or any other point on the parking lot for that matter, and self-drive to a parking lot achieving a desired final pose while avoiding obstacles on the way there.

According to the requirements, our algorithm has been designed to have two states, one is path tracking state and the other is obstacle avoidance state. For the path tracking state, the RCV follows the dynamically adjusting path created by a fuzzy logic controller based on the real-time GPS location and heading data. When an obstacle is detected by our own sonar ranging system securing the whole perimeter of the RCV, the state is switched from path tracking to obstacle avoidance. Another fuzzy logic algorithm is applied here to get around the obstacles based on the distance signals collected by 5 ultrasonic sensors. When the risk is no longer posed, the path tracking state will once again take over and lead the vehicle to the target parking lot.

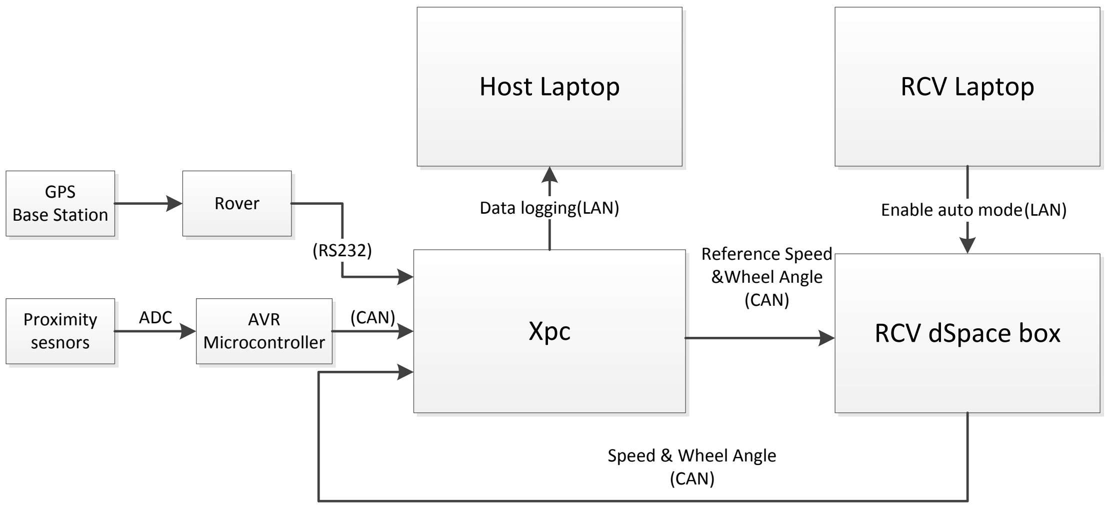
We have successfully tested the path tracking state with the RCV and achieved the real-world autonomous parking outside of the Transport Labs. Our method has been verified to be effective for the obstacle avoidance task using the Prescan simulation environment. Due to a time limitation this state has not been tested on the RCV.

What we’ve done:

- Developed the cruise control and steering control loop in dSPACE
- Developed the ultrasonic ranging system.
- Built our own coordinate system with the Trimble GPS
- Developed the algorithm based on fuzzy logic method and simulated with Prescan
- Integrated all the subsystems above.
- Successfully tested the autonomous parking in path tracking state.

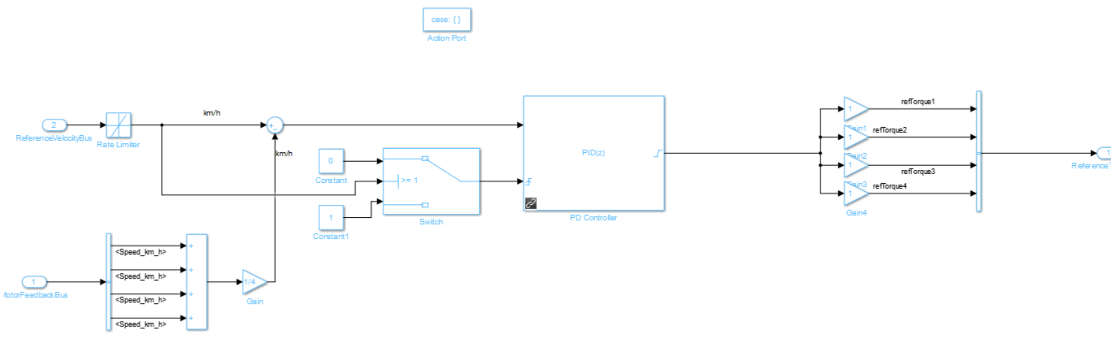
Architecture:

Here is the architecture of our system which includes three inputs and two outputs. GPS data gets into SpeedGoat (xPC) via serial port while ultrasonic sensor data is transmitted via CAN bus 1. RCV dSPACE box is connected to SpeedGoat with CAN bus 2. The receiving data is the real speed and wheel angle of RCV measured by encoders for monitoring and data logging. The sending data is the reference speed and reference wheel angle which are our control signal to RCV.



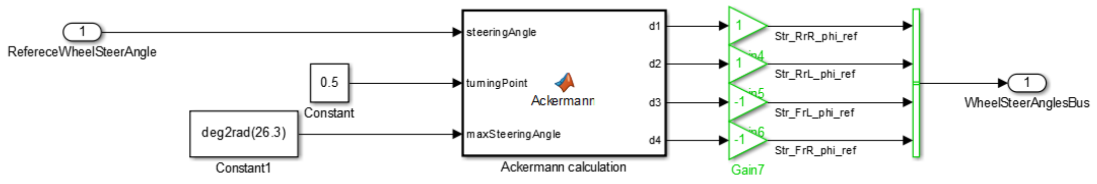
Cruise control and wheel angle control design

Cruise control and wheel angle control blocks were built in the RCV’s dSPACE box. The picture below is the cruise control block, which is a PI control mechanism. A rate limiter is placed after the reference velocity input port in order to make RCV initialize smoothly and stop immediately. The purpose of the switch block is to reset the PI controller when the reference speed is set to zero. In addition, anti-windup function in PI controller is activated so that the problem of saturation is prevented.



Cruise control loop

The input of wheel angle control ranges from -30 degree to 30 degree. Thanks to Petter’s good work, the reference wheel angle is converted into four wheel steering angles after Ackermann calculation block.



Wheel angle control loop

RCV Sonar ranging system

General information

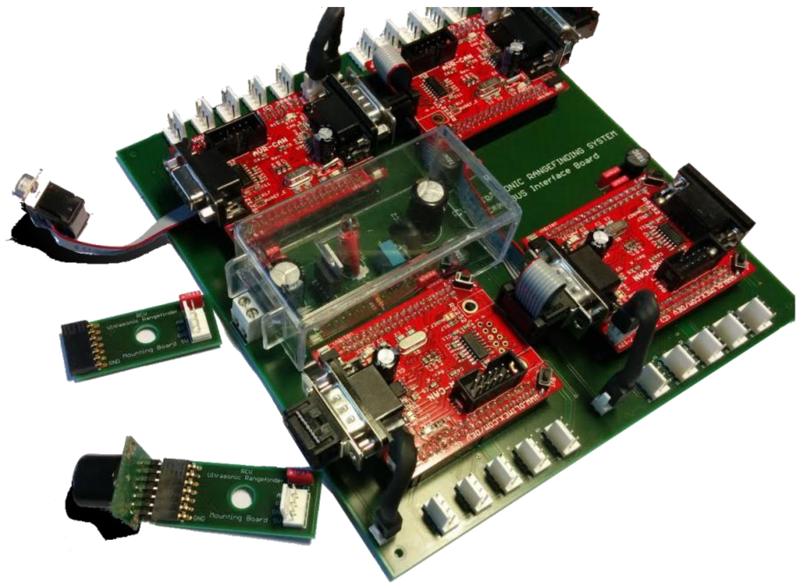
This is a sonar ranging system that uses ultrasonic sensors to measure dissonance to objects on up to 20 locations and then communicates the data via CAN BUS. It is currently designed to use MAXBOTIX sonar rangers that send an analogue voltage as their data to the micro controllers on the main PCB, but it can also be redesigned from the EAGLE CAD files and built to use sensors with other means of communication (digital) using the same idea and power supply design.
It is powered by the 24V rail on the car.

Requirements:

- OS: Windows 7 or newer, 64bit
- AVR Studio (to program the AVR controllers and change CANBUS settings)
- CAN KING for setting up and debugging

Elements

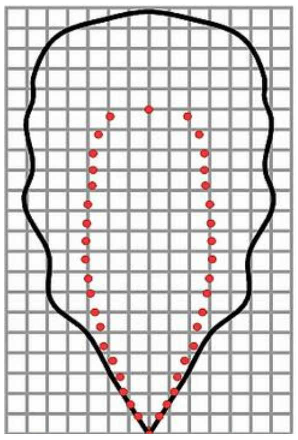
Here we will discuss the general architecture of the system.



Main PCB (Green) – it is the “motherboard” of the system, providing power and I/O to the different replaceable computing components (AVR CAN microcontrollers). It is custom designed for this specific application and if it breaks you will need to order a new PCB and build it yourself (SEK 5000). It has space for 4 AVR CANs – Front, Rear, Left and Right. You can choose any combination of them depending on your needs. Please observe the orientation of plugging the controllers according to the picture provided here. Each of the controllers has space for up to 5 sensors. The AVR code is provided in the Transport Labs repository.

AVR CAN (https://www.olimex.com/Products/AVR/Development/AVR-CAN/resources/AVR-CAN.pdf) (Red) – This controller is used to sequence the ranging sensors so that they don't interfere with each-other and to broadcast the information packet on the CAN BUS. It can be powered trough the CAN BUS too, **but it is not recommended. Always use the provided power cables to the main PCB.**

Sonar rangers (http://www.maxbotix.com/Ultrasonic_Sensors/MB1202.htm)
These are calibrated ultrasonic sensors with automatic ranging. The ones that are on the system at the time of writing this manual are the widest FOV available, though they can be exchanged with many other variations with analogue output from the same manufacturer – MAXBOTIX. They are sensitive to high moisture and water. **Do not use them in the wet!** If you need to drive the car in the aforementioned conditions please remove the sensors from the mounting PCBs



The current sensor we're using is the MB1202 model with the widest possible view angle. The problem with that is that while it is very sensitive and accurate in controlled ideal conditions, in the real world it picks up a lot of disturbances from the environment. Also, since the field of view has the shape of a very steep and wide cone, it will often detect the floor unless it is perfectly smooth. This is why we needed to tilt the sensors up away from the ground, which lowered our detection precision close to the ground. This is why we recommend using sensors with less wide a field of view if available.

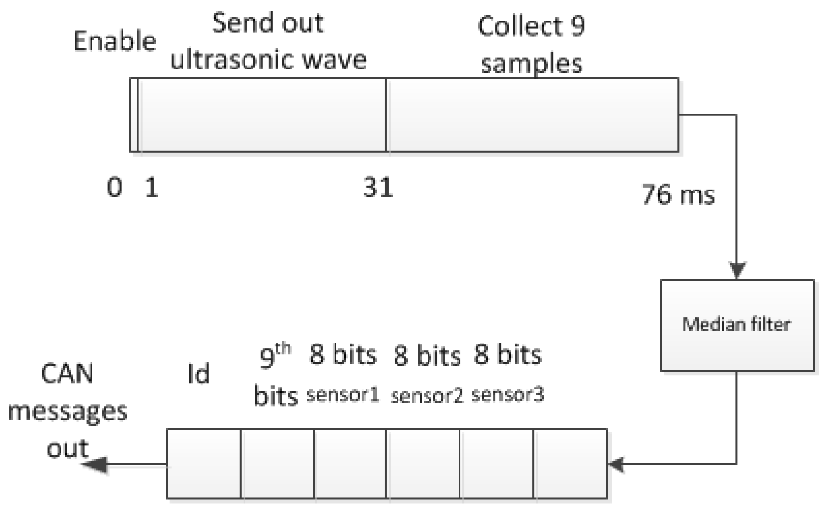
Mounting PCBs – they are custom PCBs designed to mount the sonar rangers to the car with the provided clamps, nuts and bolts and to facilitate a cable interface with the main board. They are replaceable (~25 in total for spares). Depending on the type of sensor used, please make sure the angle of mounting of the sensor w.r.t. the ground is correct for the particular senor you're using

(see sonar sensor description)

Box – the box has mounting holes for the main PCB. Mounting must happen without the micro-controllers plugged in. You must use the provided PCB stand offs in case there is any conductive dust or debris on the bottom of the case. The box has a side mounting system for the RCV seat rails and is designed to fit under the seats. Power comes from a proprietary power plug that must be attached to the main 24V battery. There is also an additional temporary output of 5V to power the 3D printed Arduino-based IMU that is on the car at the time of writing the manual.

CAN BUS Communication

Each of the controllers has its own CANBUS id and they all broadcast on the common ribbon cable. The messages they send are not synchronized between the controllers (i.e. the AVRCANs do not communicate between them) and have the following structure:



In order to make the message fit the standard 8 byte size we have implemented a shared most significant bit – if sensor #n has value above 8 bits (9 bits), the 8 least significant bits are left in the #n byte as usual, but the n-th bit of byte #9 is set to 1. The first byte is the id of the controller (F, R, L, R). It is your responsibility to check the code and make sure the ids match on the sonar ranging system side with the receiving (e.g. RCV, SpeedGoat) side.

Fuzzy Logic Algorithm:

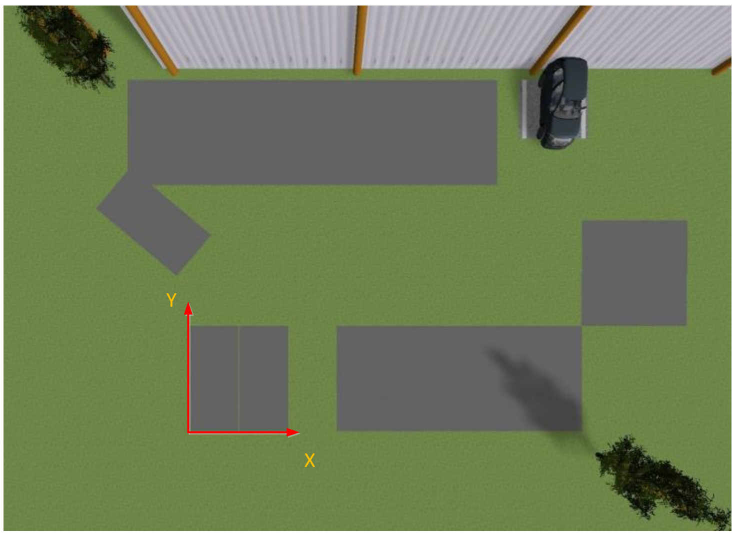
Based on the objective, our algorithm consists of two modes, one mode is path tracking state and the other one is obstacle avoidance state. In path tracking state, the car follows the path created by fuzzy logic controller to the given parking lot. When an obstacle is detected on the way to the parking lot, the path tracking state is switched to obstacle avoidance state immediately. It means another fuzzy logic controller takes over to get around the obstacles. After risk has been cleared, the path tracking state is back leading the car to the given parking lot. Considering that inertia would not let the car stop immediately, both speed and wheel angle are set to zero when the car position is within 30 cm around the target parking lot.

Path tracking state:

Inputs: X, Y, Heading;
Outputs: Steering, Speed

First of all, we built our own coordinate system as showed below. X positive direction is along with the wall of Transport Labs. Our heading is defined from -180 degree to 180 degree. Then the next step is to convert geodetic longitude and latitude into our pre-defined coordinate system.

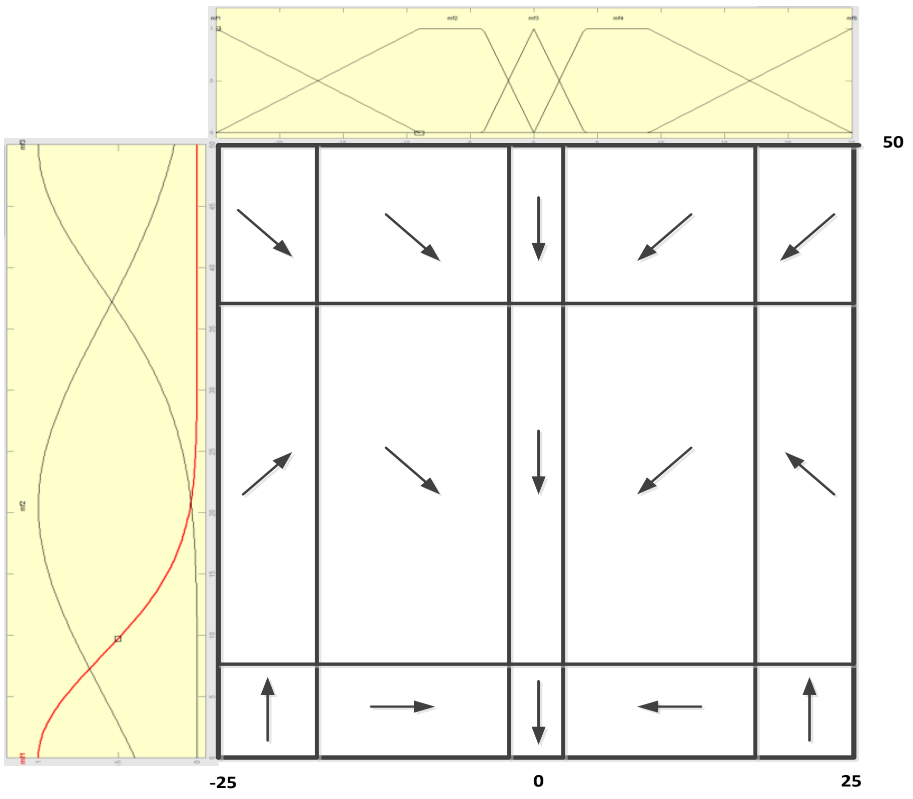
The LLA2FLAT function is used in conversion. The important factor is the clockwise degree from north to our positive X axis. With the help of Google Earth, this angle was measured approximately 56.5 degrees. As for the heading, it is easy to be solved by turning the compass heading with a certain angle, 57 degrees in our case. So far, all the inputs are ready.

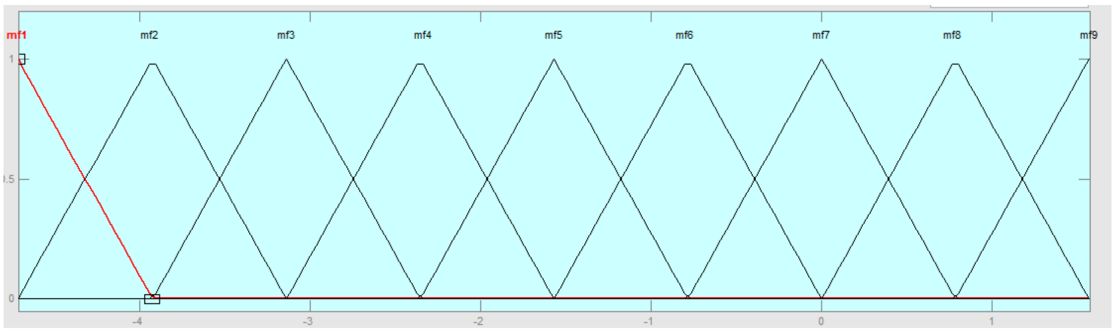


Fuzzy logic design is fuzzy and time-consuming. So our initial point was to make it as simple as possible i.e. a car always parks at the initial point of our coordinate system with the heading of -90 degree. A feasible pattern of our fuzzy logic controller has been determined after dozens of trails. The output of the fuzzy logic controller is the reference heading, which ranges from -270 degree to 90 degree. This range is strictly determinate as the difference angle of two adjacent arrows needs to be as exactly as 45 degrees e.g. combination of -90° and -45° are different from combination of 270° and -45° .

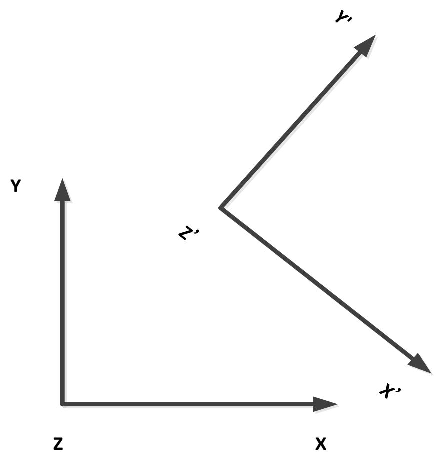
One important thing to mention is that the width of the third membership in X direction implies how accurate the final pose is. If an accurate final heading is required, then a narrow triangle is necessary. As a result, it calls a high demand of the wheel angle. If you want to know more about it, please watch the four videos in the folder of Videos and pictures-simulation videos.

“Narrow” refers that the range of the third membership in X is from -2 m to 2 m while “ wide” means the membership ranges from -4 m to 4 m. The numbers e.g. 23 and 32 are the limit value of the wheel angle. With the permutation and combination of these four variables, our simulator gives out four simulation results, from which we are able to see how important the third membership and the limit of the wheel angle are to the success of autonomous parking in our project.

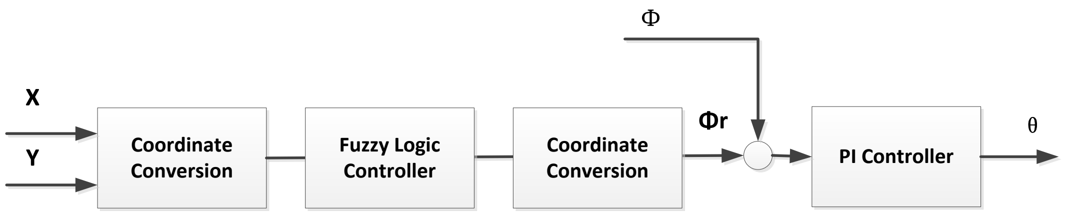




Parking at one predefined point with a certain angle is just a special case. In order to make it more general, arbitrary parking lot and pose should be taken into account. In the diagram below, z' is the chosen parking lot and –Y' is the final pose. Then a local coordinate system has been built, to which our fuzzy logic controller is possible to apply. The output of the fuzzy logic controller now is the heading in the local coordinate, which need to be converted into the global one for further calculation.

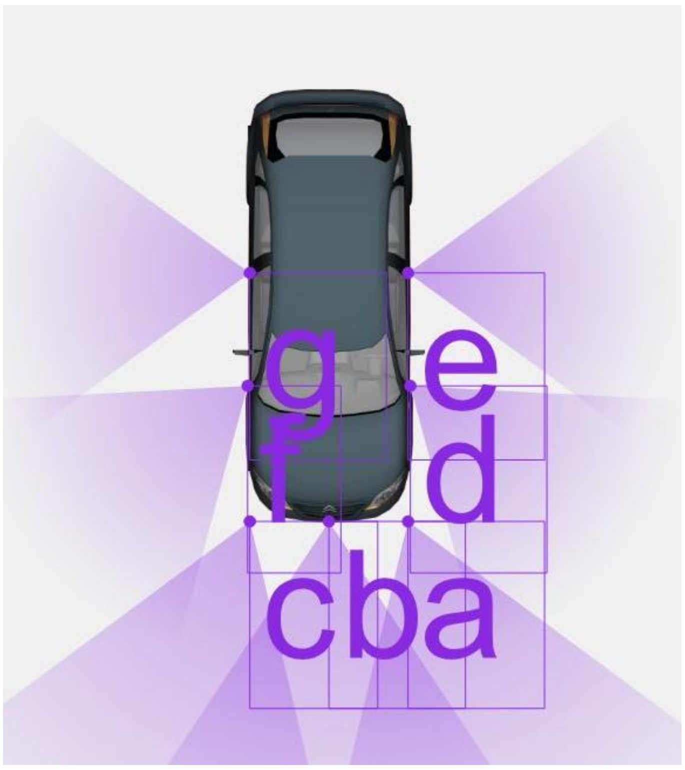


Our flow chart of path tracking state algorithm is showed as below. Φ is the real heading and θ is the wheel angle (-30 degree to 30 degree).

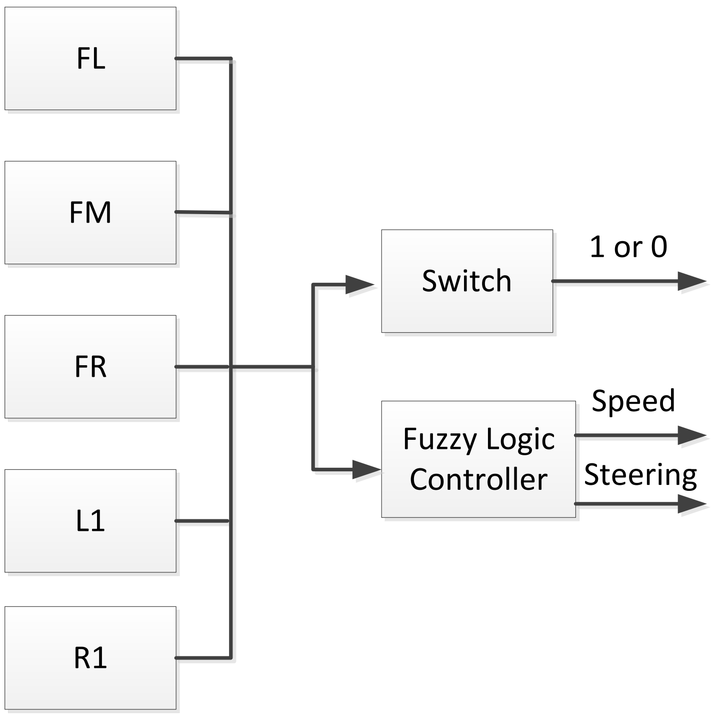


Obstacle avoidance state:

There are seven ultrasonic sensors installed in the vehicle. Five of them have been used in the simulation i.e. three in front and the front one on left/right side. When any of front sensors detects an obstacle within 4.5 m or any of side sensors detects an obstacle within 2.4 m, obstacle avoidance state is activated.



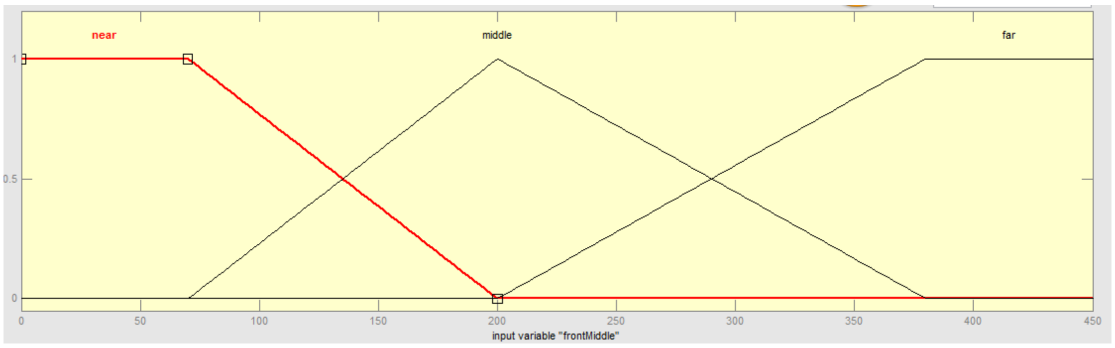
For the fuzzy logic controller, there are five inputs from five ultrasonic sensors and two outputs, speed and steering (Wheel angle).



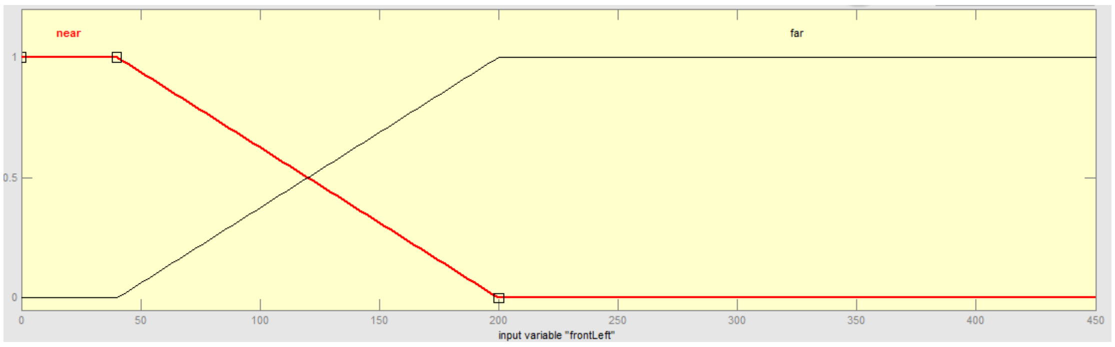
In the movement, the most important sensor is the front middle. So three memberships are assigned to it. For the rest of sensors, each has two memberships. Currently there are 48 rules (see fuzzy logic algorithm.xls) in total running in the fuzzy logic controller. Theoretically, more rules would make the movement more accurate and secure. On the other side, it would take more computation power and more time to make all rules work well. It is critical to make a balance between accuracy and complexity.

Based on the principle of symmetry, left side sensors have the same memberships as the right ones. It also applies to the front left and right ones. All the membership diagrams are shown as follows.

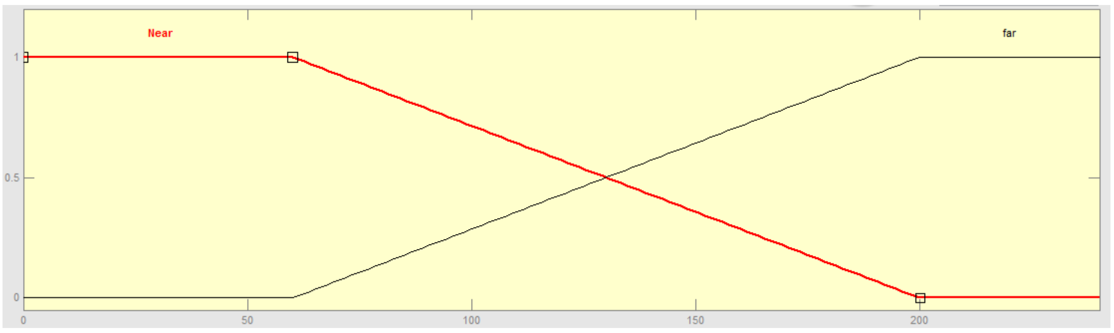
Input:



Front Middle



Front Left & Right

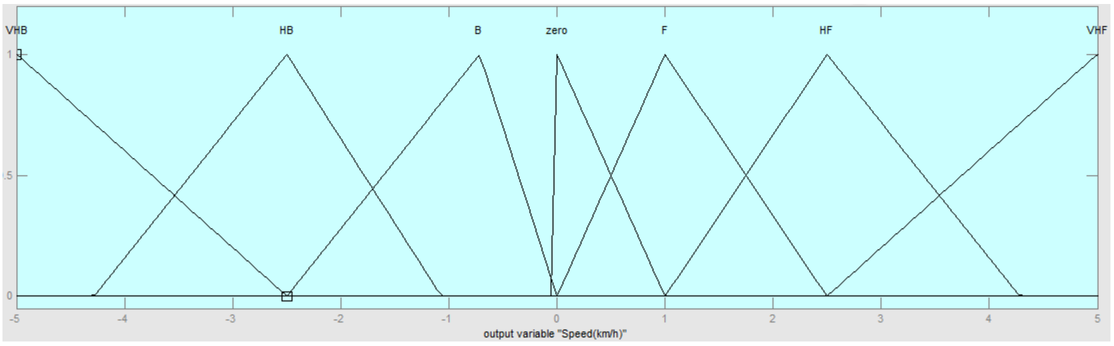


Left & Right

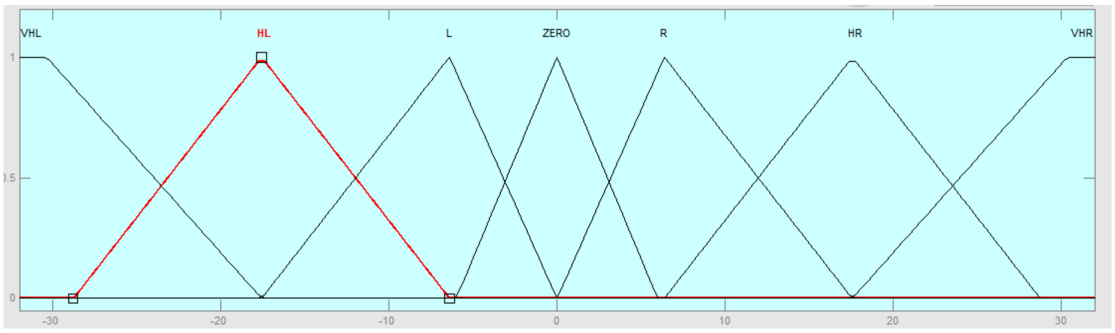
Output:

There are two outputs, speed and wheel angle, of the fuzzy logic controller. Each one has 7 memberships (VHB—very high backward, VHF—very high forward, VHL—very high left, VHR

—very high right). Speed ranges from -5 Km/h to 5 Km/h. Wheel angle ranges from -30 degree to 30 degree.

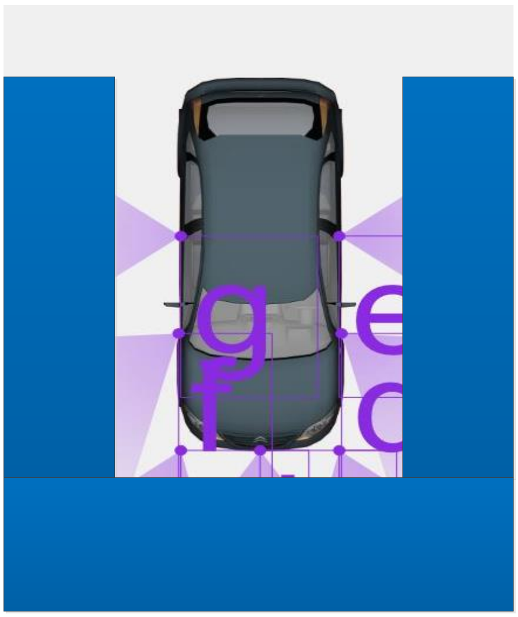


Speed



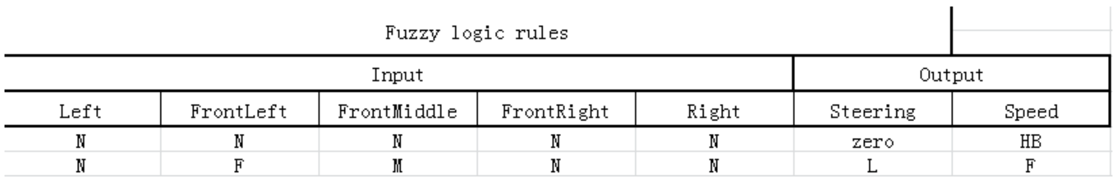
Wheel Angle

The reason why we include the backward speed is that backward motion might be a solution to the circumstance showed in the picture. A car goes in a dead end and all sensors warn that obstacles are in "Near" membership. There is no doubt that the ideal way to get out of this dilemma is go backward without steering. But this would cause some problems in some situations.

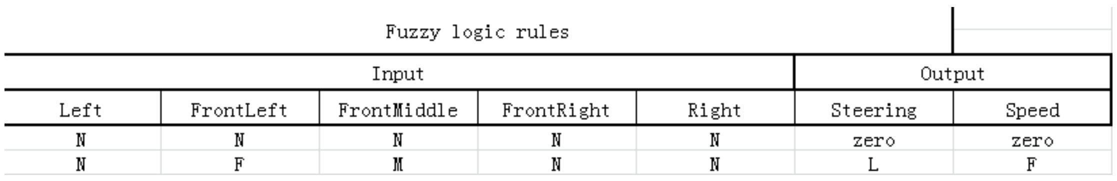


For example, if a car is standing at a certain position at the moment, two following rules are involved. One rule wants to go backward while the other intends to go forward. Different weights on rules might cause the car to stop. However, in most of our cases, there are usually more than 8 rules involving in, which have bigger chances to find a balanced point (Stop). In views of this problem, we decided to remove all the backward demand out of our fuzzy logic rule list. So the car will stop when it finds itself trapped in a dead end for the security consideration.

Two videos named "demo3" and "demo4" show how two fuzzy logic principles work in simulation. The fuzzy logic controller in the video "demo3" involves with backward mechanism. It attempted and went back many times before finding the way to get out of the dangerous area. In many unsuccessful cases, the car just stopped on the way backward. Obviously, "demo4" works much better the "demo3".



Fuzzy logic rules						
Input					Output	
Left	FrontLeft	FrontMiddle	FrontRight	Right	Steering	Speed
N	N	N	N	N	zero	HB
N	F	M	N	N	L	F



Fuzzy logic rules						
Input					Output	
Left	FrontLeft	FrontMiddle	FrontRight	Right	Steering	Speed
N	N	N	N	N	zero	zero
N	F	M	N	N	L	F

We have finished the functional simulation in Prescan. But we don't have enough time to finish the test with RCV. Undoubtedly, some rules still need to be modified according to the performance in real test. 48 rules imply that the test is doomed to be tough.

Simulation:

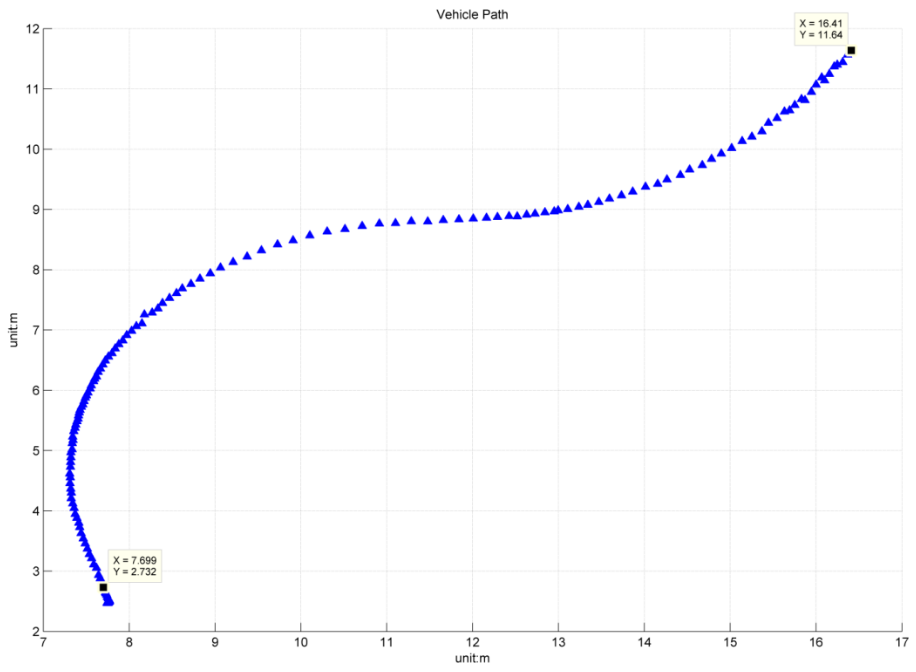
Prescan simulator in smart mobility lab is used to visualize a car's movement with the action of fuzzy logic controller. There are many types of virtual sensors that can be added in the car which would provide the source of input signals. Also, user is able to design the operation environment and make the simulation scene as similar as the real one. On the other side, the weakness of this simulator is also prominent. Firstly, the model of a car is based on the bicycle model instead of a four-wheel model. Secondly, all the sensors in Prescan are too perfect to reflect the defective performance of the sensors in real life, which caused us spending more time debugging and tuning the control parameters.

Data analysis

GPS Settings: 10Hz update rate

GPS coordinate tracking accuracy – there is very little data corruption with noise levels consistent of centimeter precision. This performance is still valid even in the beginning when the car is situated immediately next to building. Data fluctuation and outliers are kept to a very low minimum. In conclusion, the coordinate tacking of the GPS system is very reliable.

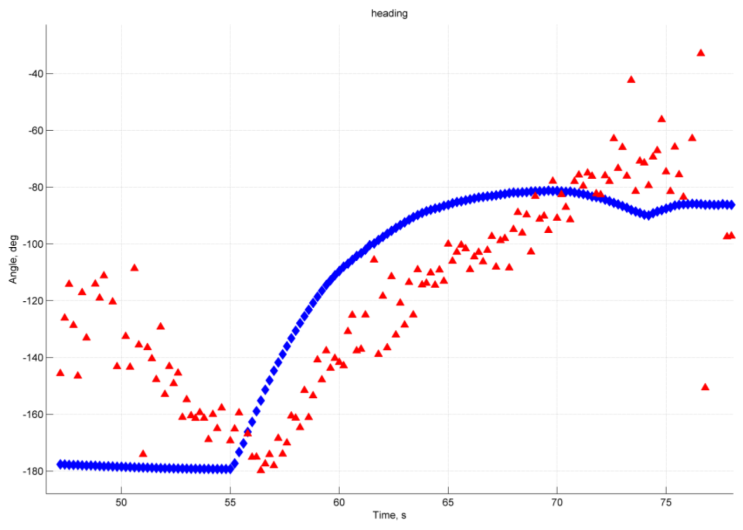
Please note the constant rate of GPS data reporting (10Hz) and the result that stems from this fact – the higher the speed, the less dense the data is. This could potentially be detrimental to performance in high-speed applications.

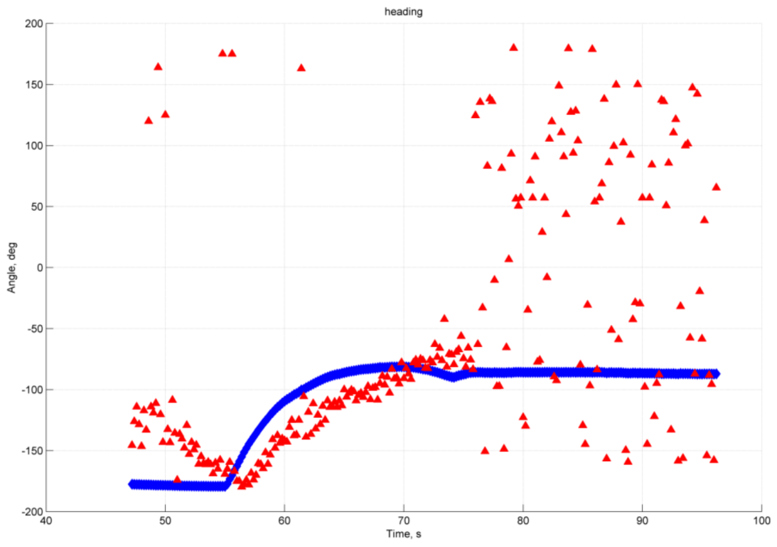


GPS heading accuracy

Even though for an ideal setup one would need 2 rover units operating in a master-slave relationship in order to create a vector of heading, we achieved a reasonably usable heading with just one rover GPS unit. The problems that arise with using GPS for heading are evident in the low speed and stationary areas of the plot below – the data is unusable under a certain speed (2 km/h). Once the speed falls below 0.5 km/h the data becomes random and needs to be filtered.

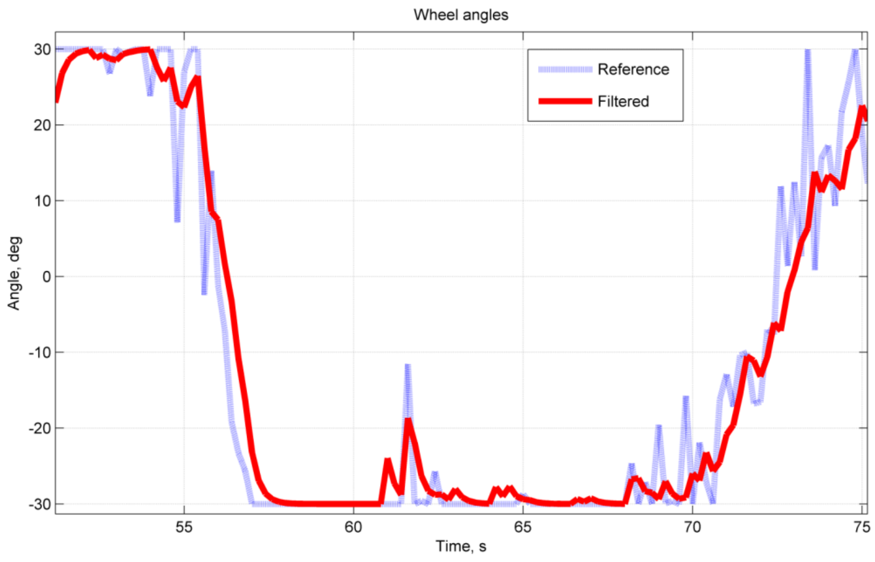
Another problem was the near proximity of buildings and trees which aided the generally low accuracy and delayed reaction to direction change. This lag was our biggest problem. Unfortunately we did not seek ways to address these problems but with model predictive control and data filtration we believe the performance could be improved. We however do not recommend using this system as a primary heading estimator in the same conditions.





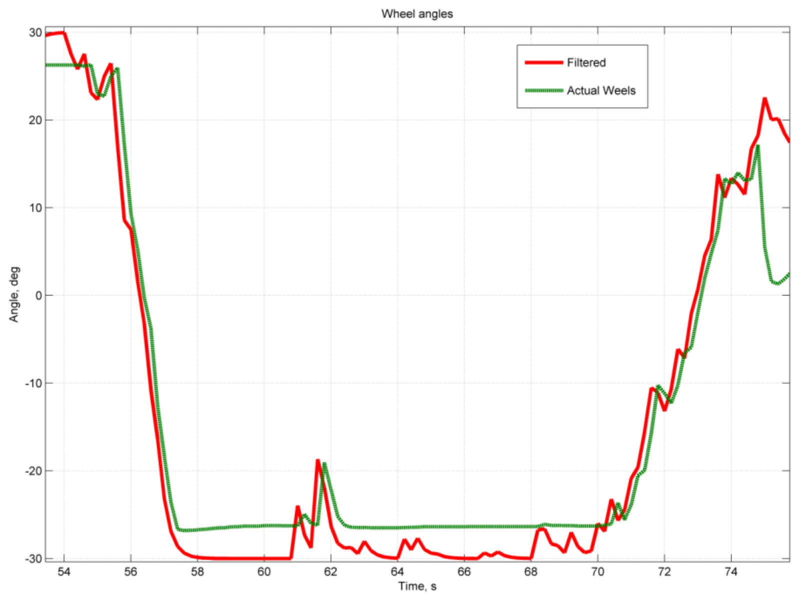
Fuzzy-logic controller output and filtered steering input

The nature of the fuzzy logic dictates that the output should be somewhat smooth compared to the relay-like step control of a simple lookup table. This however does not factor in conditions like border-line states and fluctuating heading reading, which contributed to some very sharp and dangerous heading references. This was remedied by engineering a low-pass discrete filter which removed the spikes in the signal without adding any significant lag.



Steering input and actual steering angle

Here we see the steering controller does a fantastic job of following the reference signal we assign. Of course, due to the poor low-speed steering capabilities of the RCV's suspension and actuators we were never able to reach what we estimated to be the minimum steering angle to maneuver within the desired unusually tight parking lot. Apart from that there was little-to-no lag added to the actuation by the steering controller.



Appendix:

Trimble GPS user manual

Basic information

Type: Trimble SPS 852 GNSS modular receiver
Admin: admin
Password: password
Serial number of the Lab GPS unit: 5035K69898
Serial number of the Scania GPS unit: 5039K70904

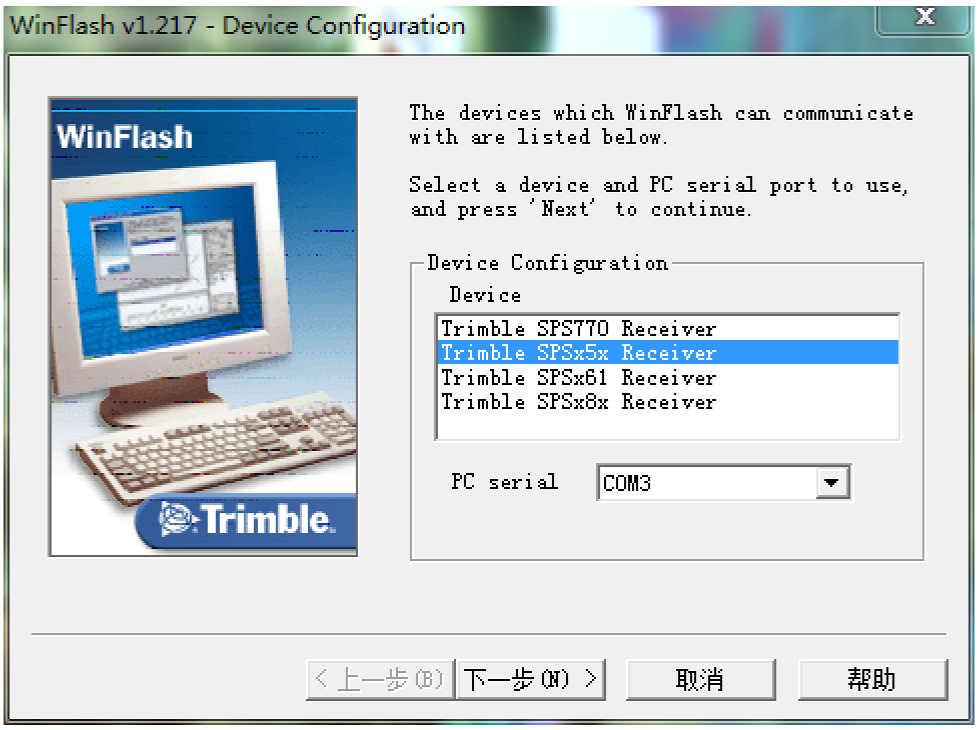
Trimble Web:
http://construction.trimble.com/products/site-positioning-systems/sps855-gnss-modular-receiver

Contact person:
Ingi from Sitech (ingi.gudmundsson@sitech-sverige.com)

Software

WinFlash, which can be downloaded from the web above, is used to upgrade the Trimble GPS by installing the license (it can do more than upgrade).

Choose "Trimble SPS x5x Receiver" and right "PC serial" port, and then click "Next". In the operations, please choose "update receiver options".


WinFlash v1.217 - Device Configuration

The devices which WinFlash can communicate with are listed below.

Select a device and PC serial port to use, and press 'Next' to continue.

Device Configuration

Device

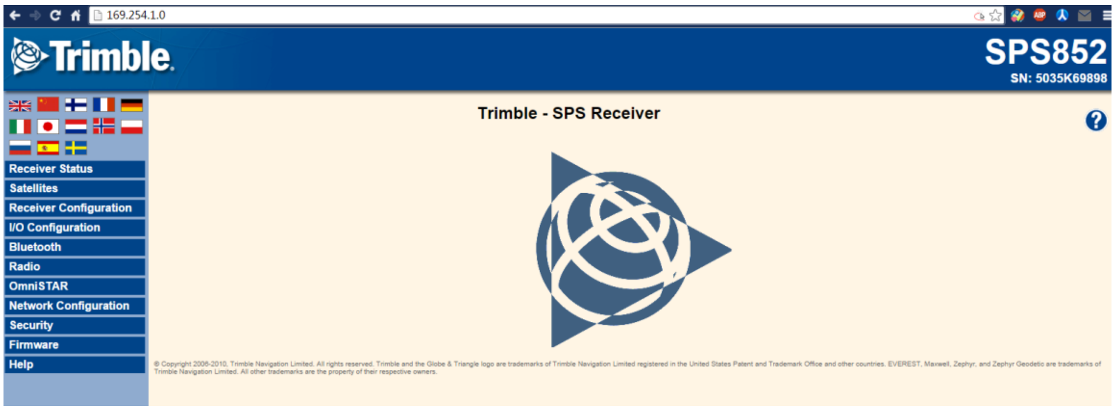
Trimble SPS770 Receiver
Trimble SPSx5x Receiver
Trimble SPSx61 Receiver
Trimble SPSx8x Receiver

PC serial: COM3

Web configuration:

Web configuration is able to do more than WinFlash can do. It is recommendable to configure the GPS units in the Web.

1. Connect a GPS unit to a PC with a LAN cable. Make sure the setting (**local area connection properties—internet protocol version 4—General—Obtain an IP address automatically**) is enabled.
2. Make sure DHCP is enabled. In the GPS interface, **Enter—Ethernet Config—DHCP**.
3. Type "**169.254.1.0**" in browser and enter the admin and password. You will see the interface as follows.



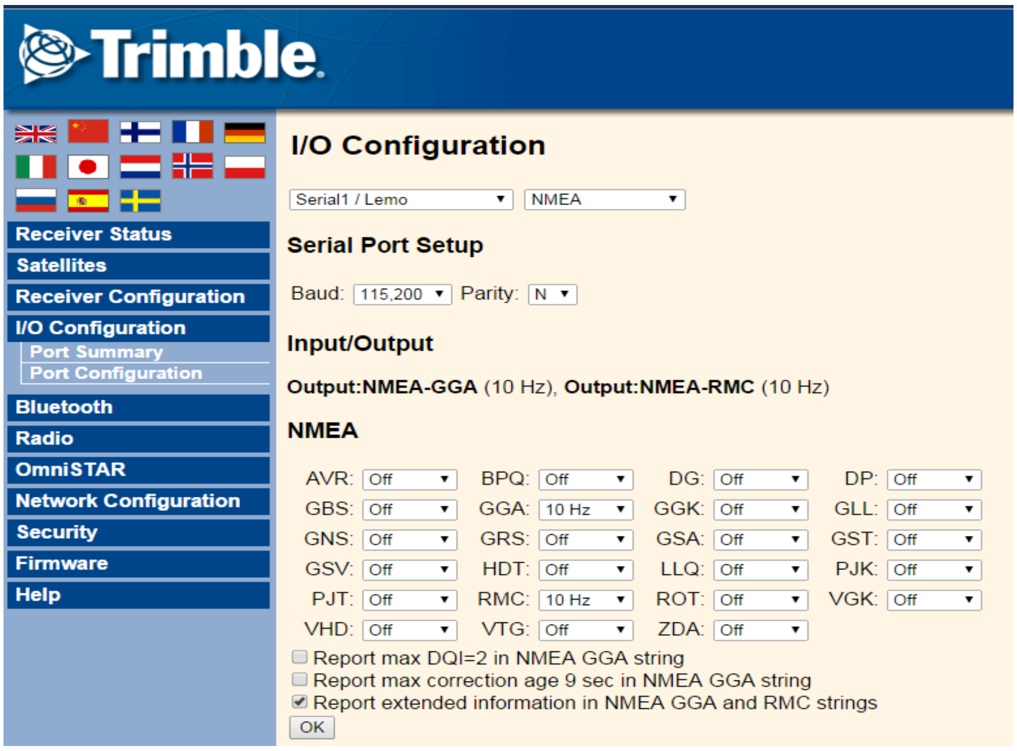
Trimble - SPS Receiver

How to save/upload a setting file

You can save your setting by downloading the application file instead of making a bunch of screenshots.
Receiver Configuration—Application Files—operation (download/upload files)—choose a file

I/O Configuration:

In our project, GPS data is transmitted via a Lemo cable. In NMEA list, you can enable the signals you want to obtain from GPS. GGA is the GPS coordinate and HDT is the heading. The meaning of other abbreviations can be found in the help session.



I/O Configuration

Serial1 / Lemo   NMEA

Serial Port Setup

Baud: 115,200 Parity: N

Input/Output

Output:NMEA-GGA (10 Hz), **Output:NMEA-RMC** (10 Hz)

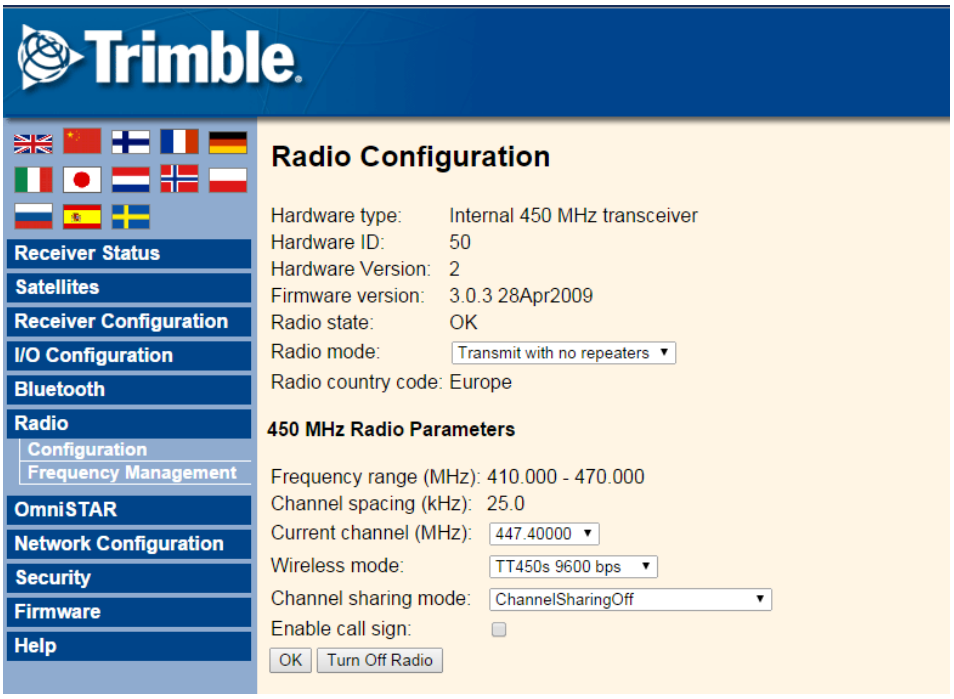
NMEA

AVR:	Off	BPQ:	Off	DG:	Off	DP:	Off
GBS:	Off	GGA:	10 Hz	GGK:	Off	GLL:	Off
GNS:	Off	GRS:	Off	GSA:	Off	GST:	Off
GSV:	Off	HDT:	Off	LLQ:	Off	PJK:	Off
PJT:	Off	RMC:	10 Hz	ROT:	Off	VGK:	Off
VHD:	Off	VTG:	Off	ZDA:	Off		

☐ Report max DQI=2 in NMEA GGA string
☐ Report max correction age 9 sec in NMEA GGA string
☑ Report extended information in NMEA GGA and RMC strings

Radio:

Make sure both Rover and Base station have the same radio frequency to communicate.



Radio Configuration

Hardware type:	Internal 450 MHz transceiver
Hardware ID:	50
Hardware Version:	2
Firmware version:	3.0.3 28Apr2009
Radio state:	OK
Radio mode:	Transmit with no repeaters
Radio country code:	Europe

450 MHz Radio Parameters

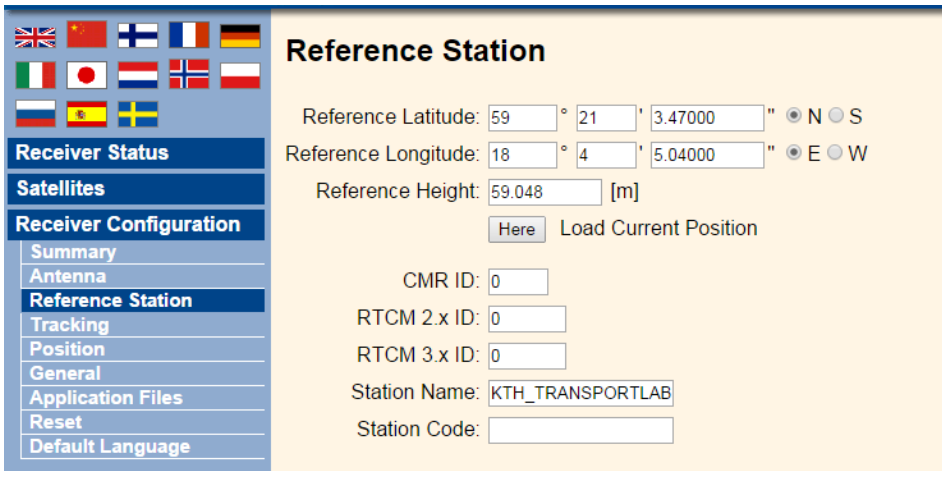
Frequency range (MHz):	410.000 - 470.000
Channel spacing (kHz):	25.0
Current channel (MHz):	447.40000
Wireless mode:	TT450s 9600 bps
Channel sharing mode:	ChannelSharingOff
Enable call sign:	☐

OK | Turn Off Radio

How to set up a reference station

For the setting file of our Base unit, you can find it in our back-up USB stick with the name of **BaseConfig**. Our base unit was set up on the roof of transport lab. Both GPS and radio antennas were connected to the GPS unit.

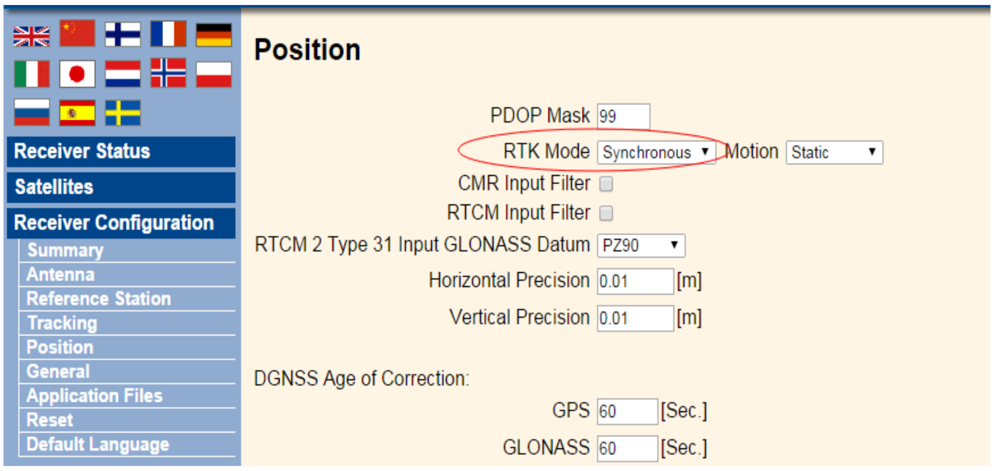
Our reference coordinate, which is obtained from Google Earth, is pinpointed at one corner of the roof. Reference height is of no importance in our case (Don’t make it too big).



Reference Station

Reference Latitude:	59 ° 21 ' 3.47000 " [✓ N] S
Reference Longitude:	18 ° 4 ' 5.04000 " [✓ E] W
Reference Height:	59.048 [m]
	Here Load Current Position
CMR ID:	0
RTCM 2.x ID:	0
RTCM 3.x ID:	0
Station Name:	KTH_TRANSPORTLAB
Station Code:	

RTK mode should be **Synchronous**. Then you will see a word of “Trans” flash on the GPS screen, which means it is sending data.



Position

PDOP Mask 99
RTK Mode Synchronous Motion Static
CMR Input Filter ☐
RTCM Input Filter ☐
RTCM 2 Type 31 Input GLONASS Datum PZ90
Horizontal Precision 0.01 [m]
Vertical Precision 0.01 [m]

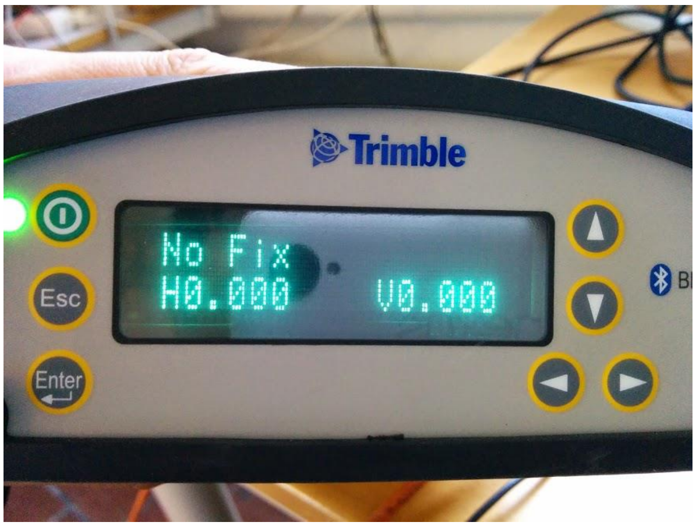
DGNSS Age of Correction:
GPS 60 [Sec.]
GLONASS 60 [Sec.]

How to set up a rover:

For the setting file of our Rover unit, you can find it in our back-up USB stick with the name of RoverConfig. Then connect two antennas to the Rover unit and you will see “Reci” flash on the GPS Screen.

Precision:

Our GPS’s theoretical precision is 7 centimeters as showed in Receiver option. Actually it is able to locate within 2 or 3 centimeters if it is fixed by satellites. As the following picture shows, it has many modes like No fix, fixed, float, Location, the meaning of which can be found in specification. If a GPS unit is placed in house or close to a building, then it would always show “float” instead of “fixed” or location. “H0.172” refers to current precision “theoretically 0.172 m”.



Real-time target computer: “Speedgoat”
User Manual

General information

This is a mobile real-time Matlab target machine manufactured by the Swiss company SpeedGoat GmbH and uses their own proprietary drivers within Matlab. It is programmed by creating a Simulink model which is then compiled and downloaded onto the SpeedGoat. Below are the detailed instructions on installation, use and debugging.

Requirements:
- OS: Windows 7 or newer, 64bit
- Matlab 2014b or newer (because it contains a stand alone code compilation mode)
- Kingston USB stick for kernel transfer
- Wired LAN connection with the host PC (for when the stand alone mode is NOT used or for data logging and monitoring purposes)

Support:
There is already a support peson which has answered many questions regarding this specific unit.

Diego Kuratli
Development Engineer and
Team Lead Quality Engineering
diego.kuratli@speedgoat.ch

Installation instructions:

In order to use the Speedgoat, Matlab needs to have the proprietary drivers from Speedgoat installed. At the time of publication the latest version for our unit is 8. To install this version 8 of the Speedgoat Tools and Drivers library, made for Simulink Real-Time and backwards compatible with xPC Target Releases R2009b-R2013b, please proceed as follows:

Prerequisites
- MATLAB with Simulink Real-Time or xPC Target license (included in most KTH licenses)
- Administrator rights for MATLAB (Run MATLAB as an Administrator)
- Compatible MEX C/C++ compiler installed and configured (for a list of compatible compilers see http://www.mathworks.com/support/compilers/). Before starting the installation you should check that your compiler is configured correctly by typing mex –setup at the MATLAB command prompt.

Installation procedure

1) Download the library from www.speedgoat.ch/downloads/sglib/speedgoat_8.zip
2) Extract the ZIP file to a temporary folder on your development computer e.g. C:\temp\speedgoat_8
3) Open MATLAB and navigate to the extracted folder.
4) Once you have found the speedgoat_setup.p file run it by typing speedgoat_setup at the command prompt.
5) Follow the instructions. The complete procedure may take few minutes.
6) Once finished, exit and restart MATLAB.

Get started

- Type speedgoat at the MATLAB command prompt to get started.
- The main driver library is accessible by typing speedgoatlib at the MATLAB prompt. The library is also available in the Simulink library browser.
- Use speedgoatkerneltransfer and the provided USB drive to configure and install a new kernel on to your Speedgoat real-time target machine. This function replaces speedgoatmachineboot from previous versions.
- You can open Speedgoat Product Documentation by typing speedgoatdoc. Additional documentation is available online at www.speedgoat.ch/support.

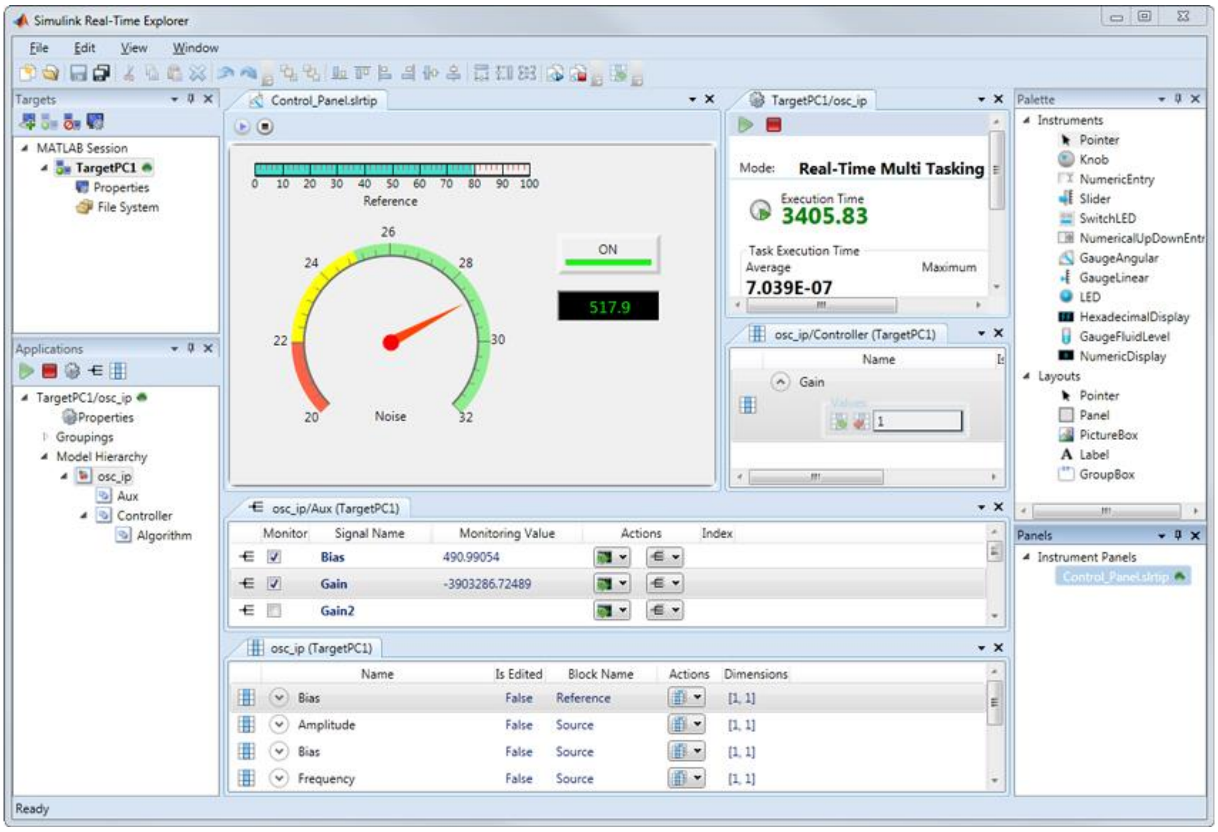
MathWorks Software Patches

R2014a kernel patch

For R2014a, Speedgoat strongly recommends to install a kernel patch resolving potential memory and graphics issues. To install please proceed as follows:
1) Close and restart MATLAB (make sure to run MATLAB as an Administrator)
2) Download the patch: www.speedgoat.ch/downloads/bugfix/MathWorks/R2014a_kernelpatch/Patch_R2014a.zip
3) Extract the ZIP file in the MATLAB workspace
4) In MATLAB, navigate to the extracted folder and locate the patch_installer.m script
5) Run patch_installer and follow the displayed instructions
6) Once the patch is installed, please restart MATLAB
7) Generate and transfer a new kernel, using speedgoatkerneltransfer. (more info on this step later)
Additional software patches required depending on specific use cases or target machine configurations only:
www.speedgoat.ch/Resources/Support/TargetMachineSetup/index.html?software_updates_and_patches_performance.htm

Establishing connection:

1) Start XPC Explorer (or Simulink Real Time Explorer in Matlab 2014b and up) with inputting "xpcexplr" in the Matlab command-window.



Mode: Real-Time Multi Tasking

Execution Time 3405.83

Task Execution Time
Average 7.039E-07
Maximum

osc_ip/Aux (TargetPC1)

Monitor	Signal Name	Monitoring Value	Actions	Index
✓	Bias	490.99054		
✓	Gain	-3903286.72489		
	Gain2			

osc_ip (TargetPC1)

Name	Is Edited	Block Name	Actions	Dimensions
Bias	False	Reference		[1, 1]
Amplitude	False	Source		[1, 1]
Bias	False	Source		[1, 1]
Frequency	False	Source		[1, 1]

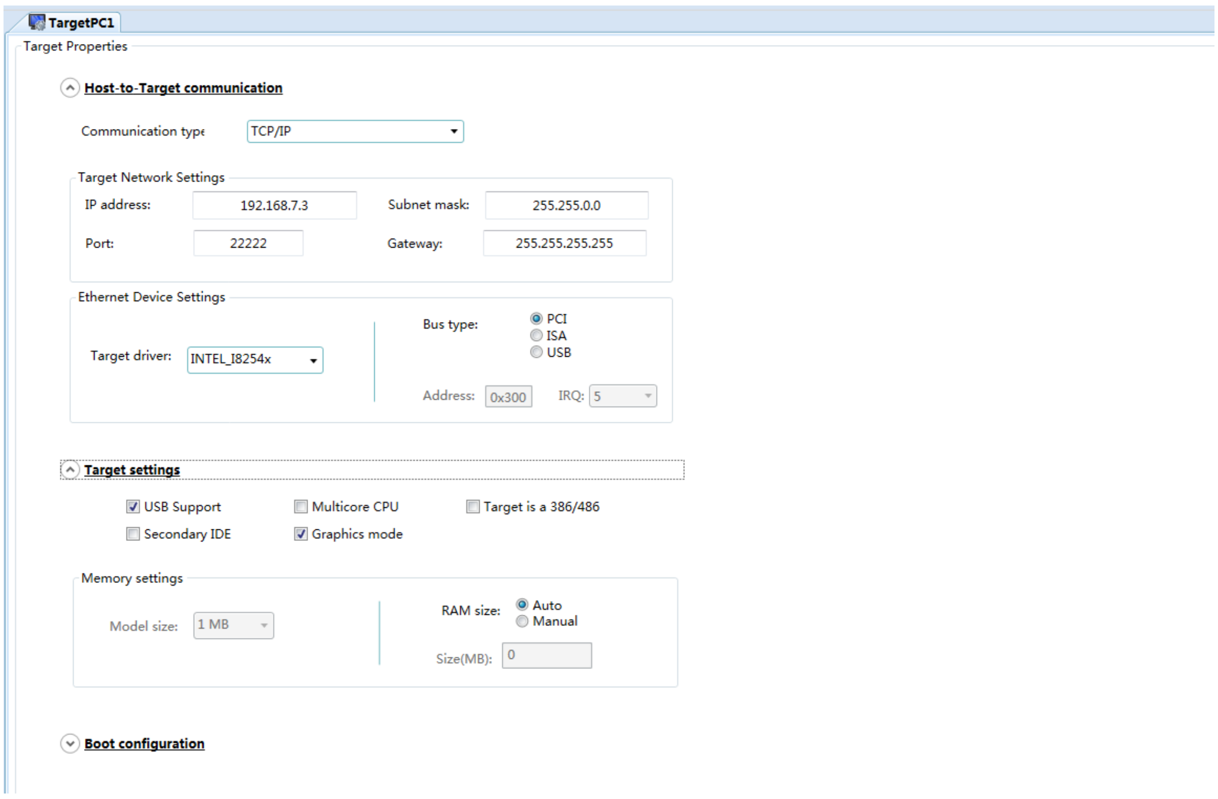
On the top-left side under "MATLAB Session" there might be a target PC already defined. If you are sure it is not used by another person delete it and add a new Target PC using the button just above. You will need to then go to the properties page by clicking the "Properties" sub-category just underneath your "Target PC1" and input the following settings:

Please keep in mind that you need to have your computer's wired network settings configured as to have your computer (Host PC) in the same LAN as the Speedgoat, but defined as a different peer, i.e. the first three numbers of the IP address should be the same, but the last number must be different for the Speedgoat and the Host PC:

Speedgoat: XXX.XXX.XXX.a

Host PC: XXX.XXX.XXX.b

For our specific case, we used:


TargetPC1

Target Properties

Host-to-Target communication

Communication type: TCP/IP

Target Network Settings

IP address:	192.168.7.3	Subnet mask:	255.255.0.0
Port:	22222	Gateway:	255.255.255.255

Ethernet Device Settings

Target driver: INTEL_I8254x

Bus type: [✓ PCI] ISA USB

Address: 0x300 IRQ: 5

Target settings

[✓] USB Support
[ ] Multicore CPU
[ ] Target is a 386/486
[ ] Secondary IDE
[✓] Graphics mode

Memory settings

Model size: 1 MB

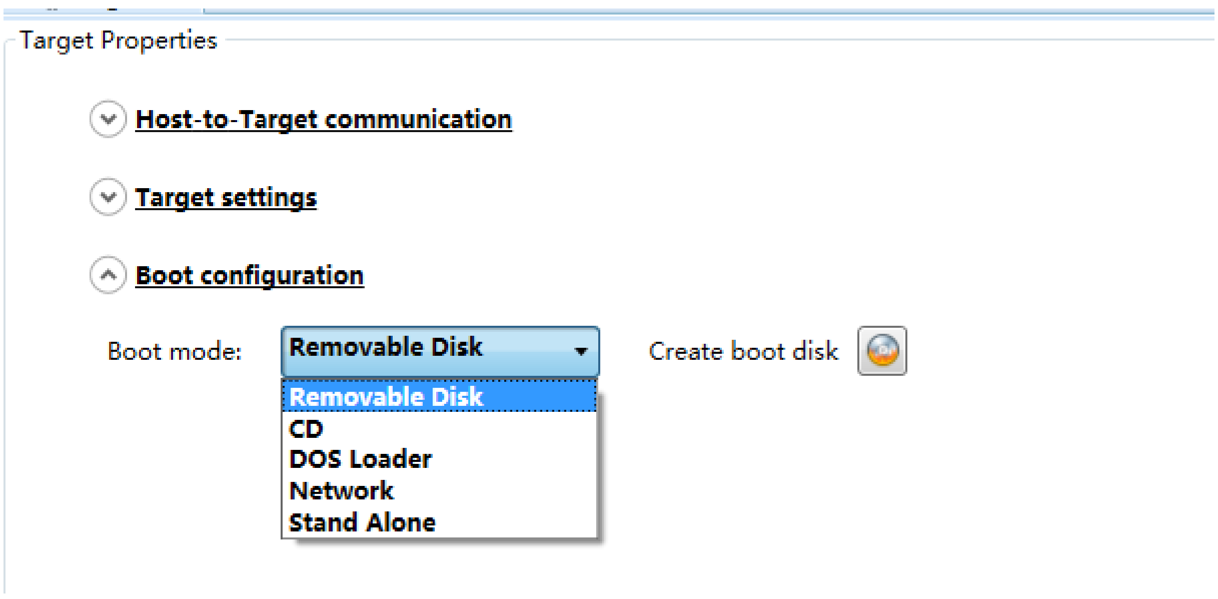
RAM size: [✓ Auto] Manual

Size(MB): 0

Boot configuration

Our host PC had its network settings modified to have an IP of 192.168.7.4.

Kernel transfer


Target Properties

Host-to-Target communication

Target settings

Boot configuration

Boot mode: Removable Disk

Removable Disk
CD
DOS Loader
Network
Stand Alone

Create boot disk

In order to use the Speedgoat as a real-time target machine it needs a kernel from which to run off. This kernel must be transferred using the specific provided Kingston USB stick which is one of the very few that are supported on this machine. When you modify the settings in xPC Explorer, make sure that you save the configuration. For boot configuration you need to either choose Removable Disk or Stand Alone. More on these later. **Do not create disk from here!** - it is for normal xPCs, the Speedgoat needs its own tools.

Then type speedgoatkerneltransfer in the command window. Use the tool to choose the appropriate setting and depending on what you have chosen you will have 2 or three files on the USB stick, for example:

- dos.sg
- xpckrnl.rtb

Please verify that these files are in the USB stick, before connecting it to the target machine.

Then, connect the USB stick into a free USB port of your target machine. During the booting time, the execution will stop at the DOS command line, showing a message "kernel transfer successful". Then, you will have to remove the USB stick and reboot the target machine. If you don't see the message, the kernel transfer failed. In that case, you can try to reboot again the target with the USB stick connected, or use another USB port.

Operation

Depending on the needs of the user, the kernel may be bare and just carry the basic networking and hardware settings done in XPC explorer (Simulink real time explorer in Matlab 2014b and up) or have the compiled code within itself. How to transfer kernels will be explained later.

<u>**Normal mode:**</u>

In the case of a bare kernel you will need to upload the Simulink program you have compiled additionally trough the LAN cable and you must do so every time the machine is started. In this case you need to have chosen and saved the option "Removable disk". It is not suitable for autonomous operation. The benefit is that you only transfer the blank kernel once and then only upload the software additionally very quickly over LAN, so it is useful for prototyping. After the machine has been successfully connected you can either have it uploaded automatically when the compilation of the code trough the “Build model” button in Simulink finishes, or you may sometimes get an error saying that it cannot find the target. This is often nothing to worry about, and means you need to upload the code manually. This is done by right-clicking on the real-time machine in your Simulink real-time Browser and clicking “Upload” , then choosing the .dlm file you have just compiled from the model. This could also be done if you want to change the code with another one you have already compiled at a different time.

Standalone mode:

If you need to have the Speedgoat somewhere where you will leave it to operate autonomously and will not make changes very often (e.g. in a car) you can use the stand alone mode. In this case you need to have chosen and saved the option “Standalone”. In this mode the kernel has the compiled code embedded in itself and thus the when the Speedgoat is started it begins code execution immediately and without the need for a host computer to start it manually. The disadvantage is that it is very clumsy and slow to transfer kernels every time you make small changes and thus should be used when the code is ready for longer periods of testing.

Data logging:

Data logging is done by having a File Scope in your model. Make sure you have a large enough sample size. It starts when the program is run so there might be a lot of unnecessary data in the beginning if you are using the standalone mode. The code for data extraction is on the KTH Transport Labs file server. The code stops the execution of the program and converts the data of some or all file scopes to a readable structure in your work space. Please note you might need to change the code according to your needs. **Do not power off the Speedgoat before you extract the data. Depending on your settings you might lose it!**

Sample data logging extraction code (save as a *.m file and run):

```
%% Code

addpath(genpath(fullfile(pwd,'GCDC2012-Library'))); %Very important library for datalogging.

%Please make sure you have this in your MATLAB path.
```

```
%% Implementation:
tg = xpc;
stop(tg)                                        % Stop xPC Target Object (run the model)
% Wait until model has finished running.
while ~strcmpi(tg.Status,'stopped');      % Is the run complete?
end;
% Get data.
%tg = xpc;
for i=tg.Scopes(1:end)                          % Get file scope (Id: 2)
```

```
        sc = getscope(tg,i);
        if strcmpi(sc.Type,'File');

                fsys = xpctarget.fs;                              % Connect to the target PC file system
                h = fsys.fopen(sc.FileName);                      % Open the log file
                fsysData = fsys.fread(h);                         % Read the data
                fsys.fclose(h);                                   % Close the log file
                current_name=sc.FileName(1:end-4);
                Results.(genvarname(current_name)) = readxpcfile(fsysData);      % Convert uint8 log data to double
and log under Scope name
                Results.(genvarname(current_name)).sc=sc;
                Results.(genvarname(current_name)).tg=tg;
                clear fsys fsysData current_name h sc
        end
end
```

Debugging:

Please note the Speedgoat can only use I/O blocks from its own library in SMULINK, thus limiting your choice, which is bot a good and a bad thing. On the plus side you have a smaller and easier choice, but on the downside, these blocks may have bugs which you will bring with them.

Problems and solutions:

1) In the Autonomous RCV model files the CAN BUS message unpack block always reads the first element in the message as 0 no matter how long (bitwise) it was or how many other elements there were. We therefore had to modify what the RCV sent to go around this problem. Another way to fix this is to use the old "obsolete" blocks from version 2, but in other cases other bugs may appear.

2) Another problem is the simulated real-time – it means that for every iteration the execution time must be lower than the discrete time you set in the model. That means that some delayed blocks may cause "CPU overflow" error message. This is either fixed by using an optimized block, loosening the sample time of the model or disabling the real-time check (not recommended!) What we did was to find that the task execution time is 0.12 ms while our fix-step of Simulink model is 0.1 ms. This is done in the property window of xPC target. The overload problem was addressed after we changed our fix-step to 0.2 ms.

3) When we monitored the signals obtained from dSPACE box (the main computer of the RCV), we found that the steering angle signal responds 3 seconds after we turn the steering wheel. The reason was that most of the CAN messages in dSPACE CAN bus are sent out every 30 ms. When

the sending time arrives, those messages are send out in a certain order according to the priority. The time interval between two different messages is less than 0.2 ms. When the Receive and Read Module retrieves our target messages in intervals of 0.2 ms, it cannot guarantee that the target message is captured all the time. That’s the source of the delay. There are two solutions. One is to decrease the retrieving period of the Receive and Read Module to 0.1 ms or even lower, but this was incompatible with our previous solution to increase the sampling time to 0.2ms. So we tuned the RCV's message broadcasting frequency for the messages we were interested in different from that of the other messages.