# Software

Concrete fixes die je al kan toepassen als je niet de hele codebase weggooid:

* Zorg dat de software niet afhankelijk is van IP adressen of andere constante waardes. Bijv: vision IP moet op runtime aanpasbaar zijn, robot COM poort moet op runtime aanpasbaar zijn, etc. Je moet niet hoeven recompilen als je je laptop van het veld naar de tafel verplaatst (waar je ineens in de simulatie verder moet oid).

Make the software independent from static values. For example, make the IP addresses from the vision cameras runtime configurable. Same goes for the COM port for the base station ('/dev/ttyACM0'). You shouldn't need to recompile if you change a few constants. It should be runtime configurable

* Door buffering van ROS queues is het moeilijk om echt 60Hz te halen in de control loop. Maar: queues zijn wel fijn voor debugging (afluisteren messages e.d.). Overweeg gedeeltelijk over te stappen op een polling i.p.v. messaging architecture.

Because of the ROS queue buffering, it is hard to reach 60Hz within the control loop. But: queues are fine for debugging. Think about using polling instead of messaging architecture

* We hebben dit jaar vaak last gehad van de “pinguin bug”. Dit is een misleidende naam, want het is eigenlijk een symptoom. Dit gebeurt wanneer een robot buitensporig veel robot commands genereert, i.e. alle robots draaien op 60Hz, behalve robot 5, die op 500Hz draait. Dit zorgt ervoor dat geen enkele robots genoeg berichtjes binnen krijgt aangezien ze allemaal verdrinken.
  + Oplossing 1: maak het statisch checkbaar dat er per robot maar 1 berichtje per cycle gegenereerd wordt. Is misschien mogelijk met een linear type oid.
  + Oplossing 2 (wordt sws aangeraden): schakel over op meerdere topics voor robot commands. Op dit moment komen alle robot commands aan op /robotcommands. Maak er in plaats daarvan per robot 1, i.e. /robot0commands, /robot1commands, etc. Op die manier als robot 3 flipt verdrinken de andere commands niet. Dit heeft als bijkomend voordeel dat je de queue sizes van elk robotcommands topic op 1 kan zetten, waardoor altijd alleen het allernieuwste pakketje verzonden wordt.

The "pinguin bug" is actually a symptom. All robots post on the topic /robotcommands. When this happens, a robot generates an excessive amount of messages, at around 500Hz, drowning out all the other messages. Because of this, the robots don't receive enough messages.

**Fix 1** : check that each robot only generates one message per cycle. Might be possible with a linear type or something.

**Fix 2** : (recommended) move from one topic to multiple topics. Instead of one topic /robotcommands, move to /robot0commands, /robot1commands, etc. If one robots drowns, it won't block the messages from the other robots. Additionaly, the queue of each topic can be set to 1, assuring that only the latest commands are sent.

* Enable c++14 and refactor code
* Replace the constants generation thing in roboteam\_utils (or maybe just roboteam\_msgs?) with just a standard ros message with a bunch of constants in it that are properly named according to some convention, i.e. topic names are of form TOPIC\_X, PARAM\_X, PARAM\_X\_DEFAULT, etc. I would not put nodenames in there, as no code should depend on node names. (there are a few exceptions, and there are bad ones).
* Protobuffers generation in roboteam\_utils should probably put the sources in the lib folder and not the include folder
* We cannot play on an assymetric field; it is a assumed the field is symmetric and the goals are not flipped. This is a dirty hack to not have to switch names around when you scale field elements (arcs and lines).
* LastRef/LastWorld (in both singleton and bt param form), should be const refs. Right now we are creating and copying them all over the place which results in Uniformly Slow Code. Talking mostly about LastWorld::get() here, which returns a value. This should be a const ref. Then again, at every callsite of LastWorld::get(), the variable assigned to should also be a const ref. (auto const & should do the trick).
* Behavior3:
  + Deleting default parameters from a node (in the left sidebar) somehow renames every instance of that node throughout the b3 project
* default ID type throughout the codebase. right now we just use int/unsigned int/size\_t mixing and matching to satisfy the compiler. It should just be an unsigned int typedef., robotID.
* We hebben in onze codebase de functies findBot/getWorldBot. Natuurlijk mag er maar 1tje bestaan.
* debug\_print seems kind of sucky. causes recompiles all over the place. having all the debug knobs in one file though is handy at times.
  + 31 august 2017: you definitely want to get rid of this abomination. ROS has debugging facilities (ROS\_DEBUG\_NAMED bijv.) die exact doen wat debug\_print moet doen, maar dan 10 x beter.

Minder concrete fixes (vraag evt. oude team om toelichting, dit zijn allemaal brainfarts):

* Skills don't compose (altough it is possible with a few guidelines). Undesirable.
* Testability is horrible not that horrible, just unclear. (can be improved though; see below). I think you also want to be able to unit test the output of the robot commands of some skills. I.e.: update should return besides a status also a robot command, or something similar? Can in theory be done with an external listener
  + (actually not that bad. there are a few global things which are ugly, but nothing's stopping you from setting them in a proper test harness)
* Compile time is horrible. Is this because of ROS? We use some boost as well (which is well-known for its compile times) and rely on headers to trigger recompilation of generated trees. Altough most of the compilation time is not really spent in trees I think.
* Barrier of entry with C++ is very high. Recode in Java with ROS as a conceptual blueprint? Or look intro ROS java?
* Multiple repo's seems like fake decoupling and hinders code sharing (if we want to make a controller in robothub using code from tactics is very hard). But it also seems to work somewhat (i.e. keyboard controller decoupling is nice).
  + Do we even want code sharing between repo’s? Why make a controller in robothub?
* Usability is also hindered by global variables (lastWorld et al.): you can never really be quite sure what all the stuff is that some piece of code means.
  + However, while debugging it was never really a problem. Each program just had to make sure that it was set properly. Since our number of programs < 100, this was doable.
  + possible solution: tree library should have template parameter (RTT\_Context) in which all the lastworld stuff is located (and global publishers). That way you can check with the compiler whether or not you are passing enough information, and it is unit testable.
  + I like this. Nice and functional. Clear. You can even make stuff optional in the rtt\_context struct or provide fallbacks.
* ROS hinders adoption on windows? Java would be ideal. Unless ROS starts supporting windows! (they are in fact working on this)
* If they implement their own messages, incorparate an option/sum type as well
* Remove boost. This might be hard, but really we only really use filesystem and optional. So in theory we could build our own version of optional and compartmentalize filesystem to one specific source file. This could help compile times.
* Some values should be "global"
  + Example: our\_field\_side
    - vision forgets to initialize it, defaults to left
    - rolenode forgets to initialize it, defaults to right
    - error!
  + there should be some authority over which variable defaults to what?
  + Probably one class that’s shared across the codebase would work. However that would hurt modularity. Maybe through a message with constants?
  + Best option: see the concrete fixes above.
* Wss spul herschrijven in python ipv shell?
* we need a stricter separation between "control land" and "skill land"; i.e. you want to share control between skills, but you don't want to nest skills (makes things confusing to debug) -> control should be in pure routines!
  + Bonus: this makes it wayyy easier to unit test control too! If this even something to be unit tested.
* (june 20th) it seems that ros publisher buffering prevents real-time 60hz for world updates. e.g. world publishes a msg, it's buffered, world publishes another msg, two messages are sent by ros, rolenode receives 2 messages but only keeps one, ergo 30 fps instead of 60. unsure if asynchronicity can be disabled; rospy suggests yes, roscpp suggests no.
  + Seems like the only way to avoid this is to run \*everything\* in one process. Since then intraprocess messages are used, which means when something is published it's directly pushed in a queue, and not sent over TCP first.
  + Or nodelets: <http://wiki.ros.org/nodelet> too much for this year probably though.
  + <http://answers.ros.org/question/264207/optionally-disable-ros-publisher-buffering/> see my question
* merge conflicts in behavior trees zijn eng en daardoor moet je vet veel bomen kopieren als je wil samenwerken aan een boom. Dus idealiter is er een mergetool voor het json formaat.

Create a merge tool to solve merge conflicts in the JSON behaviour trees

* strategy composer refactoren naar een ding dat een mapping als argument krijgt een een (optional) strategy tree teruggeeft. voor elke stage heb je een aparte mapping van refcommands naar strategy trees.
  + Ik dacht dat dit nodig was tijdens het kampioenschap, maar t is prima gelukt zonder. gecombineerd met het feit dat dit een best zware datastructuur zou zijn lijkt me dit niet nodig.
* need a way to see what the robot is "thinking". stdout works, but becomes crowded when multiple things happen and is hard to keep clean/throw old things out when they are not needed anymore (i.e. gotopos within getball within role tree). some kind of multimedia display seems like a good idea (i.e. a canvas on which you can display some text and maybe some graphs/figures).
  + Right now we have rqt\_world\_view, which is a very good start (we have used it extensively the past year), but it’s slow/buggy and can be hard to use from the code. It should be easy if you just want to print or visuale some stuff: you shouldn’t have to do that with just a bunch of lines.
* trees need types!
  + first kind of type, the amount of robots it acquires. then you can typecheck it's neighbours and see if there are any spots where there are robots leftover.
  + So the type of a tactic is Tactic(N), where N is the amount of robots it needs.
  + also for nesting. i.e. skill type, so you can use different control processes in getball for example. (i.e. GoToPos(AgressivePID) or something) or that a paralleltactic can only have tactics as children (and/or other parallel tactics?)
* lifetime per robot per world, i.e. hoelang voor t laatst gezien.
  + Ik weet niet meer precies wat ik hiermee bedoelde, maar geloof iets dat je in de world kan zien wanneer hij voor t laatst gezien is zodat je kan zien hoe serieus je de positie moet nemen. Dit is nuttig omdat world spul filtert, dus als de robot 0,2 sec. verdwijnt houdt world hem er nog wel even in.

Implement time since the last update of the robot from the world