Textual versus Graphical Programming Languages [closed]

Asked 16 years, 4 months ago Modified 11 years, 3 months ago

Viewed 10k times

35

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 4 years ago.

Improve this question

I am part of a high school robotics team, and there is some debate about which language to use to program our robot. We are choosing between C (or maybe C++) and LabVIEW. There are pros for each language.

C(++):

- Widely used
- Good preparation for the future (most programming positions require text-based programmers.)
- We can expand upon our C codebase from last year

- Allows us to better understand what our robot is doing.

LabVIEW

- Easier to visualize program flow (blocks and wires, instead of lines of code)
- Easier to teach (Supposedly...)
- "The future of programming is graphical." (Think so?)
- Closer to the Robolab background that some new members may have.
- Don't need to intimately know what's going on. Simply tell the module to find the red ball, don't need to know how.

This is a very difficult decision for us, and we've been debating for a while. Based on those pros for each language, and on the experience you've got, **what do you think the better option is?** Keep in mind that we aren't necessarily going for pure efficiency. We also hope to prepare our programmers for a future in programming.

Also:

- **Do you think that graphical languages such as LabVEIW are the future of programming?**
- **Is a graphical language easier to learn than a textual language?** I think that they should be about equally challenging to learn.

- Seeing as we are partailly rooted in helping people learn, **how much should we rely on prewritten modules, and how much should we try to write on our own?** ("Good programmers write good code, great programmers copy great code." But isn't it worth being a good programmer, first?)

Thanks for the advice!

Edit: I'd like to emphasize this question more: The team captain thinks that LabVIEW is better for its ease of learning and teaching. **Is that true?** I think that C could be taught just as easily, and beginner-level tasks would still be around with C. I'd really like to hear your opinions. **Is there any reason that typing while{} should be any more difficult than creating a "while box?" Isn't it just as intuitive that program flows line by line, only modified by ifs and loops, as it is intuitive that the program flows through the wire, only modified by ifs and loops!?**

Thanks again!

Edit: I just realized that this falls under the topic of "language debate." I hope it's okay, because it's about what's best for a specific branch of programming, with certain goals. If it's not... I'm sorry...

robotics **labview** **graphical-language**

Share

Improve this question

Follow

edited Mar 6, 2010 at 15:26

Earlz

63.7k ● 100 ● 311 ● 506

asked Aug 17, 2008 at 15:39

stalepretzel

15.9k ● 23 ● 78 ● 92

Back when I worked for the Jet Propulsion Laboratory in 1990, LabView and "Graphical Programming" were "the future". Apparently, they still are. – Curt Hagenlocher Aug 17, 2008 at 17:25

Note that Java will also be an option this year (2010). My team used LabVIEW last year, and I hated it. We're going to be using C(++?) for sure this year. – Sophie Alpert Oct 2, 2009 at 0:57

+1 for the Graphical vs Textual programming question – thirdy Nov 13, 2012 at 4:23

My feedback on quite different field than robotic. In one of my mission in finance world, I have been forced to use some visual programming tool. This was not labView, but an EAI (a tool that integrate other pieces of software together). This EAI was a visual tool, with boxes connected together. For the main 'roads' it was fine, but the details were very ugly: logic was buried and we had to click plenty of time to analyze the 'real' program flow. It turned into visual spaghettis, with dozens of boxes and connectors around. Maintenance was a nightmare: no refactoring, no automated tests... – Guillaume Feb 18, 2014 at 10:32

25 Answers

Sorted by: Highest score (default)

35

Before I arrived, our group (PhD scientists, with little programming background) had been trying to implement a LabVIEW application on-and-off for nearly a year. The code was untidy, too complex (front and back-end) and most importantly, did not work. I am a keen programmer but had never used LabVIEW. With a little help from a LabVIEW guru who could help translate the textual progamming paradigms I knew into LabVIEW concepts it was possible to code the app in a week. The point here is that *the basic coding concepts still have to be learnt, the language, even one like LabVIEW, is just a different way of expressing them*.

LabVIEW is great to use for what it was originally designed for. i.e. to take data from DAQ cards and display it on-screen perhaps with some minor manipulations in-between. However, programming *algorithms* is no easier and I would even suggest that it is more difficult. For example, in most procedural languages execution order is generally followed line by line, using pseudo mathematical notation (i.e. `y = x*x + x + 1`) whereas LabVIEW would implement this using a series of VI's which don't necessarily follow from each other (i.e. left-to-right) on the canvas.

Moreover programming as a career is more than knowing the technicalities of coding. Being able to effectively ask for help/search for answers, write readable code and work with legacy code are all key skills which are

undeniably more difficult in a graphical language such as LabVIEW.

I believe some aspects of graphical programming may become mainstream - the use of sub-VIs perfectly embodies the 'black-box' principal of programming and is also used in other language abstractions such as Yahoo Pipes and the Apple Automator - and perhaps some future graphical language will revolutionise the way we program but LabVIEW itself is not a massive paradigm shift in language design, we still have `while, for, if` flow control, typecasting, event driven programming, even objects. If the future really will be written in LabVIEW, C++ programmer won't have much trouble crossing over.

As a postcript I'd say that C/C++ is more suited to robotics since the students will no doubt have to deal with embedded systems and FPGAs at some point. Low level programming knowledge (bits, registers etc.) would be invaluable for this kind of thing.

@mendicant Actually LabVIEW is used a lot in industry, especially for control systems. Granted NASA unlikely use it for on-board satellite systems but then software developement for space-systems is a whole different ball game...

Share Improve this answer Follow

edited Mar 6, 2010 at 15:18

answered Aug 17, 2008 at 17:23

Brendan
19.3k • 19 • 89 • 114

FPGAs can be designed in either VHDL/Verilog (textual) or schematic (graphical). Additionally you can interchange the two, so FPGA designers have the best of both worlds. It might be useful to expand on the FPGAs reference in your answer. – earlNameless Mar 9, 2011 at 18:17

12

I've encountered a somewhat similar situation in the research group I'm currently working in. It's a biophysics group, and we're using LabVIEW all over the place to control our instruments. That works absolutely great: it's easy to assemble a UI to control all aspects of your instruments, to view its status and to save your data.

And now I have to stop myself from writing a 5 page rant, because for me LabVIEW has been a nightmare. Let me instead try to summarize some pros and cons:

Disclaimer I'm not a LabVIEW expert, I might say things that are biased, out-of-date or just plain wrong :)

LabVIEW pros

- Yes, it's **easy to learn**. Many PhD's in our group seem to have acquired enough skills to hack away within a few weeks, or even less.
- **Libraries**. This is a major point. You'd have to carefully investigate this for your own situation (I

don't know what you need, if there are good LabVIEW libraries for it, or if there are alternatives in other languages). In my case, finding, e.g., a good, fast charting library in Python has been a major problem, that has prevented me from rewriting some of our programs in Python.

- Your school may already have it installed and running.

LabVIEW cons

- It's perhaps *too* easy to learn. In any case, it seems no one really bothers to learn best practices, so programs quickly become a complete, irreparable mess. Sure, that's also bound to happen with text-based languages if you're not careful, but IMO it's much more difficult to do things right in LabVIEW.
- There tend to be **major issues in LabVIEW with finding sub-VIs** (even up to version 8.2, I think). LabVIEW has its own way of knowing where to find libraries and sub-VIs, which makes it very easy to completely break your software. This makes large projects a pain if you don't have someone around who knows how to handle this.
- **Getting LabVIEW to work with version control is a pain**. Sure, it can be done, but in any case I'd refrain from using the built-in VC. Check out LVDiff for a LabVIEW diff tool, but don't even think about merging.

(The last two points make working in a team on one project difficult. That's probably important in your case)

- This is personal, but I find that many algorithms just don't work when programmed visually. **It's a mess**.
    - One example is stuff that is strictly sequential; that gets cumbersome pretty quickly.
    - It's difficult to have an overview of the code.
    - If you use sub-VI's for small tasks (just like it's a good practice to make functions that perform a small task, and that fit on one screen), you can't just give them names, but you have to draw icons for each of them. That gets very annoying and cumbersome within only a few minutes, so you become very tempted *not* to put stuff in a sub-VI. It's just too much of a hassle. Btw: making a really good icon can take a professional hours. Go try to make a unique, immediately understandable, recognizable icon for every sub-VI you write :)
- You'll have carpal tunnel within a week. Guaranteed.
- @Brendan: hear, hear!

Concluding remarks

As for your "should I write my own modules" question: I'm not sure. Depends on your time constraints. Don't spend time on reinventing the wheel if you don't have to. It's too easy to spend days on writing low-level code and then

realize you've run out of time. If that means you choose LabVIEW, go for it.

If there'd be easy ways to combine LabVIEW and, e.g., C++, I'd love to hear about it: that may give you the best of both worlds, but I doubt there are.

But make sure you and your team spend time on learning best practices. Looking at each other's code. Learning from each other. Writing usable, understandable code. And having fun!

And please forgive me for sounding edgy and perhaps somewhat pedantic. It's just that LabVIEW has been a *real* nightmare for me :)

Share Improve this answer

Follow

answered Aug 17, 2008 at 19:44

onnodb

5,261 ● 1 ● 33 ● 41

2 LabVIEW can merge VIs as of 8.5 - zone.ni.com/devzone/cda/tut/p/id/6212 8.5 also has tools to prevent crosslinking, the finding sub-vis issue you mentioned - zone.ni.com/devzone/cda/tut/p/id/6200 – Chris Hamons Dec 9, 2009 at 5:57

9

I think the choice of LabVIEW or not comes down to whether you want to learn to program in a commonly used language as a marketable skill, or just want to get stuff done. LabVIEW enables you to Get Stuff Done very quickly and productively. As others have observed, it doesn't magically free you from having to understand what you're doing, and it's quite possible to create an unholy mess if you don't - although anecdotally, the worst examples of bad coding style in LabVIEW are generally perpetrated by people who are experienced in a text language and refuse to adapt to how LabVIEW works because they 'already know how to program, dammit!'

That's not to imply that LabVIEW programming isn't a marketable skill, of course; just that it's not as mass-market as C++.

LabVIEW makes it extremely easy to manage different things going on in parallel, which you may well have in a robot control situation. Race conditions in code that should be sequential shouldn't be a problem either (i.e. if they are, you're doing it wrong): there are simple techniques for making sure that stuff happens in the right order where necessary - chaining subVI's using the error wire or other data, using notifiers or queues, building a state machine structure, even using LabVIEW's sequence structure if necessary. Again, this is simply a case of taking the time to understand the tools available in LabVIEW and how they work. I don't think the gripe about having to make subVI icons is very well directed; you can

very quickly create one containing a few words of text, maybe with a background colour, and that will be fine for most purposes.

'Are graphical languages the way of the future' is a red herring based on a false dichotomy. Some things are well suited to graphical languages (parallel code, for instance); other things suit text languages much better. I don't expect LabVIEW and graphical programming to either go away, or take over the world.

Incidentally, I would be very surprised if NASA *didn't* use LabVIEW in the space program. Someone recently described on the Info-LabVIEW mailing list how they had used LabVIEW to develop and test the closed loop control of flight surfaces actuated by electric motors on the Boeing 787, and gave the impression that LabVIEW was used extensively in the plane's development. It's also used for real-time control in the Large Hadron Collider!

The most active place currently for getting further information and help with LabVIEW, apart from National Instruments' own site and forums, seems to be LAVA.

Share Improve this answer

Follow

answered Oct 2, 2008 at 10:56

nekomatic

6,274 ● 1 ● 22 ● 27

This doesn't answer you question directly, but you may want to consider a third option of mixing in an interpreted language. Lua, for example, is already used in the

6

robotics field. It's fast, light-weight and can be configured to run with fixed-point numbers instead of floating-point since most microcontrollers don't have an FPU. Forth is another alternative with similar usage.

It should be pretty easy to write a thin interface layer in C and then let the students loose with interpreted scripts. You could even set it up to allow code to be loaded dynamically without recompiling and flashing a chip. This should reduce the iteration cycle and allow students to learn better by seeing results more quickly.

I'm biased **against** using visual tools like LabVIEW. I always seem to hit something that doesn't or won't work quite like I want it to do. So, I prefer the absolute control you get with textual code.

Share Improve this answer

Follow

answered Aug 17, 2008 at 15:55

Judge Maygarden

27.5k ● 9 ● 83 ● 100

6

LabVIEW's other strength (besides libraries) is **concurrency**. It's a dataflow language, which means that the runtime can handle concurrency for you. So if you're doing something highly concurrent and don't want to have to do traditional synchronization, LabVIEW can help you there.

The future doesn't belong to graphical languages as they stand today. It belongs to whoever can come up with a representation of dataflow (or another concurrency-

friendly type of programming) that's as straightforward as the graphical approach is, but is also parsable by the programmer's own tools.

Share Improve this answer

Follow

answered Nov 26, 2008 at 14:12

BryCoBat

677 ● 4 ● 10

4

There is a published study of the topic hosted by National Instruments:

A Study of Graphical vs. Textual Programming for Teaching DSP

It specifically looks at LabVIEW versus MATLAB (as opposed to C).

Share Improve this answer

Follow

answered Aug 17, 2008 at 16:18

Judge Maygarden

27.5k ● 9 ● 83 ● 100

8 It might be worth pointing out that NI is the makers of LabVIEW. :) – unwind Dec 9, 2008 at 9:57

4

I think that graphical languages wil always be limited in expressivity compared to textual ones. Compare trying to communicate in visual symbols (e.g., REBUS or sign language) to communicating using words.

For simple tasks, using a graphical language is usually easier but for more intricate logic, I find that graphical languages get in the way.

Another debate implied in this argument, though, is declarative programming vs. imperative. Declarative is usually better for anything where you really don't need the fine-grained control over how something is done. You *can* use C++ in a declarative way but you would need more work up front to make it so, whereas LABView is designed as a declarative language.

A picture is worth a thousand words but if a picture represents a thousand words that you don't need and you can't change that, then in that case a picture is worthless. Whereas, you can create thousands of pictures using words, specifying every detail and even leading the viewer's focus explicitly.

Share Improve this answer

Follow

answered Aug 17, 2008 at 21:34

Mark Cidade

99.8k ● 33 ● 229 ● 237

4

LabVIEW lets you get started quickly, and (as others have already said) has a massive library of code for doing various test, measurement & control related things.

The single biggest downfall of LabVIEW, though, is that you lose all the tools that programmers write for themselves.

Your code is stored as VIs. These are opaque, binary files. This means that your code really isn't yours, it's LabVIEW's. You can't write your own parser, you can't write a code generator, you can't do automated changes via macros or scripts.

This **sucks** when you have a 5000 VI app that needs some minor tweak applied universally. Your **only** option is to go through every VI manually, and heaven help you if you miss a change in one VI off in a corner somewhere.

And yes, since it's binary, you can't do diff/merge/patch like you can with textual languages. This does indeed make working with more than one version of the code a horrific nightmare of maintainability.

By all means, use LabVIEW if you're doing something simple, or need to prototype, or don't plan to maintain your code.

If you want to do real, maintainable programming, use a textual language. You might be slower getting started, but you'll be faster in the long run.

(Oh, and if you need DAQ libraries, NI's got C++ and .Net versions of those, too.)

Share Improve this answer

Follow

answered Nov 26, 2008 at 3:38

BryCoBat

677 ● 4 ● 10

4

My first post here :) be gentle ...

I come from an embedded background in the automotive industry and now i'm in the defense industry. I can tell you from experience that C/C++ and LabVIEW are really different beasts with different purposes in mind. C/C++ was always used for the embedded work on microcontrollers because it was compact and compilers/tools were easy to come by. LabVIEW on the other hand was used to drive the test system (along with test stand as a sequencer). Most of the test equipment we used were from NI so LabVIEW provided an environment where we had the tools and the drivers required for the job, along with the support we wanted ..

In terms of ease of learning, there are many many resources out there for C/C++ and many websites that lay out design considerations and example algorithms on pretty much anything you're after freely available. For LabVIEW, the user community's probably not as diverse as C/C++, and it takes a little bit more effort to inspect and compare example code (have to have the right version of LabVIEW etc) ... I found LabVIEW pretty easy to pick up and learn, but there a nuisances as some have mentioned here to do with parallelism and various other things that require a bit of experience before you become aware of them.

So the conclusion after all that? I'd say that BOTH languages are worthwhile in learning because they really

do represent two different styles of programming and it is certainly worthwhile to be aware and proficient at both.

Share Improve this answer

Follow

edited Oct 18, 2009 at 6:26

Ether

53.9k ● 13 ● 90 ● 161

answered Oct 2, 2009 at 0:39

user146093

2

Oh my God, the answer is so simple. Use **LabView**.

I have programmed embedded systems for 10 years, and I can say that without at least a couple months of infrastructure (very careful infrastructure!), you will not be as productive as you are on day 1 with **LabView**.

If you are designing a robot to be sold and used for the military, go ahead and start with C - it's a good call.

Otherwise, use the system that allows you to try out the most variety in the shortest amount of time. That's **LabView**.

Share Improve this answer

Follow

answered Aug 22, 2008 at 0:47

Frank Krueger

70.9k ● 48 ● 164 ● 211

I love LabVIEW. I would highly recommend it especially if the other remembers have used something similar. It

2

takes a while for normal programmers to get used to it, but the result's are much better if you already know how to program.

C/C++ equals manage your own memory. You'll be swimming in memory links and worrying about them. Go with LabVIEW and make sure you read the documentation that comes with LabVIEW and watch out for race conditions.

Learning a language is easy. Learning how to program is not. This doesn't change even if it's a graphical language. The advantage of Graphical languages is that it is easier to visual what the code will do rather than sit there and decipher a bunch of text.

The important thing is not the language but the programming concepts. It shouldn't matter what language you learn to program in, because with a little effort you should be able to program well in any language. Languages come and go.

Share Improve this answer

Follow

answered Aug 22, 2008 at 1:05

Moe Tsujimoto

2

Disclaimer: I've not witnessed LabVIEW, but I have used a few other graphical languages including WebMethods Flow and Modeller, dynamic simulation languages at university and, er, MIT's Scratch :).

My experience is that graphical languages can do a good job of the 'plumbing' part of programming, but the ones I've used actively get in the way of algorithmics. If your algorithms are very simple, that might be OK.

On the other hand, I don't think C++ is great for your situation either. You'll spend more time tracking down pointer and memory management issues than you do in useful work.

If your robot can be controlled using a scripting language (Python, Ruby, Perl, whatever), then I think that would be a much better choice.

Then there's hybrid options:

If there's no scripting option for your robot, and you have a C++ geek on your team, then consider having that geek write bindings to map your C++ library to a scripting language. This would allow people with other specialities to program the robot more easily. The bindings would make a good gift to the community.

If LabVIEW allows it, use its graphical language to plumb together modules written in a textual language.

Share Improve this answer

Follow

answered Nov 26, 2008 at 14:51

slim

41.2k ● 14 ● 99 ● 128

1

I think that graphical languages might be the language of the future..... for all those adhoc MS Access developers out there. There will always be a spot for the purely textual coders.

Personally, I've got to ask what is the real fun of building a robot if it's all done for you? If you just drop a 'find the red ball' module in there and watch it go? What sense of pride will you have for your accomplishment? Personally, I wouldn't have much. Plus, what will it teach you of coding, or of the (very important) aspect of the software/hardware interface that is critical in robotics?

I don't claim to be an expert in the field, but ask yourself one thing: Do you think that NASA used LabVIEW to code the Mars Rovers? Do you think that anyone truly prominent in robotics is using LabView?

Really, if you ask me, the only thing using cookie cutter things like LabVIEW to build this is going to prepare you for is to be some backyard robot builder and nothing more. If you want something that will give you something more like industry experience, build your own 'LabVIEW'-type system. Build your own find-the-ball module, or your own 'follow-the-line' module. It will be far more difficult, but it will also be way more cool too. :D

Share Improve this answer Follow

answered Aug 17, 2008 at 15:59

mendicant
27 • 4

1

You're in High School. How much time do you have to work on this program? How many people are in your group? Do they know C++ or LabView already?

From your question, I see that you know C++ and most of the group does not. I also suspect that the group leader is perceptive enough to notice that some members of the team may be intimidated by a text based programming language. This is acceptable, you're in high school, and these people are *normies*. I feel as though normal high schoolers will be able to understand LabView more intuitively than C++. I'm guessing most high school students, like the population in general, are scared of a command line. For you there is much less of a difference, but for them, it is night and day.

You are correct that the same concepts may be applied to LabView as C++. Each has its strengths and weaknesses. The key is selecting the right tool for the job. LabView was *designed for this kind of application*. C++ is much more generic and can be applied to many other kinds of problems.

I am going to recommend LabView. Given the right hardware, you can be up and running almost out-of-the-box. Your team can spend more time *getting the robot to*

do what you want, which is what the focus of this activity should be.

Graphical Languages are not the future of programming; they have been one of the choices available, created to solve certain types of problems, for many years. The future of programming is layer upon layer of abstraction away from machine code. In the future, we'll be wondering why we wasted all this time programming "semantics" over and over.

how much should we rely on prewritten modules, and how much should we try to write on our own? You shouldn't waste time reinventing the wheel. If there are device drivers available in Labview, use them. You can learn a lot by copying code that is similar in function and tailoring it to your needs - you get to see how other people solved similar problems, and have to interpret their solution before you can properly apply it to your problem. If you blindly copy code, chances of getting it to work are slim. You have to be good, even if you copy code.

Best of luck!

Share Improve this answer

Follow

answered Sep 15, 2008 at 16:28

Arc the daft

223 ● 1 ● 2

I would suggest you use LabVIEW as you can get down to making the robot what you want to do faster and

1

easier. LabVIEW has been designed with this mind. OfCourse C(++) are great languages, but LabVIEW does what it is supposed to do better than anything else. People can write really good software in LabVIEW as it provides ample scope and support for that.

Share Improve this answer

Follow

answered Sep 16, 2008 at 19:16

Manoj

5,087 ● 14 ● 55 ● 77

1

There is one huge thing I found negative in using LabVIEW for my applications: Organize design complexity. As a physisist I find Labview great for prototyping, instrument control and mathematical analysis. There is no language in which you get faster and better a result then in LabVIEW. I used LabView since 1997. Since 2005 I switched completely to the .NET framework, since it is easier to design and maintain.

In LabVIEW a simple 'if' structure has to be drawn and uses a lot of space on your graphical design. I just found out that many of our commercial applications were hard to maintain. The more complex the application became, the more difficult it was to read.

I now use text laguages and I am much better in maintaining everything. If you would compare C++ to LabVIEW I would use LabVIEW, but compared to C# it does not win

Share Improve this answer

Follow

answered Oct 24, 2008 at 20:23

Enrico

2,037 ● 3 ● 31 ● 40

1

As allways, it depends.

I am using LabVIEW since about 20 years now and did quite a large kind of jobs, from simple DAQ to very complex visualization, from device controls to test sequencers. If it was not good enough, I for sure would have switched. That said, I started coding Fortran with punchcards and used a whole lot of programming languages on 8-bit 'machines', starting with Z80-based ones. The languages ranged from Assembler to BASIC, from Turbo-Pascal to C.

LabVIEW was a major improvement because of its extensive libraries for data acqusition and analysis. One has, however, to learn a different paradigma. And you definitely need a trackball ;-))

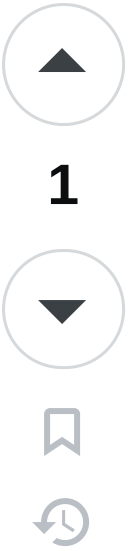
Share Improve this answer

Follow

answered Dec 9, 2008 at 9:22

Lul

49 ● 1



I don't anything about LabView (or much about C/C++), but..

> Do you think that graphical languages such as LabVEIW are the future of programming?

No...

> Is a graphical language easier to learn than a textual language? I think that they should be about equally challenging to learn.

Easier to learn? No, but they *are* easier to explain and understand.

To explain a programming language you have to explain what a variable is (which is surprisingly difficult). This isn't a problem with flowgraph/nodal coding tools, like the LEGO Mindstroms programming interface, or Quartz Composer..

For example, in this is a fairly complicated LEGO Mindstroms program - it's very easy to understand what is going in... but what if you want the robot to run the `INCREASEJITTER` block 5 times, then drive forward for 10 seconds, then try the INCREASEJITTER loop again? Things start getting messy very quickly..

Quartz Composer is a great exmaple of this, and why I don't think graphical languages will ever "be the future"

It makes it very easy to really cool stuff (3D particle effects, with a camera controlled by the average brightness of pixels from a webcam).. but incredibly difficult to do easy things, like iterate over the elements from an XML file, or store that average pixel value into a file.

> Seeing as we are partailly rooted in helping people learn, how much should we rely on prewritten modules, and how much should we try to write on our own? ("Good programmers write good code, great programmers copy great code." But isn't it worth being a good programmer, first?)

For learning, it's so much easier to both explain and understand a graphical language..

That said, I would recommend a specialised text-based language language as a progression. For example, for graphics something like Processing or NodeBox. They are "normal" languages (Processing is Java, NodeBox is Python) with very specialised, easy to use (but absurdly powerful) frameworks ingrained into them..

Importantly, they are very interactive languages, you don't have to write hundreds of lines just to get a circle onscreen.. You just type `oval(100, 200, 10, 10)` and press the run button, and it appears! This also makes them very easy to demonstrate and explain.

More robotics-related, even something like LOGO would be a good introduction - you type "FORWARD 1" and the turtle drives forward one box.. Type "LEFT 90" and it turns 90 degrees.. This relates to reality very simply. You can visualise what each instruction will do, then try it out and confirm it really works that way.

Show them shiney looking things, they will pickup the useful stuff they'd learn from C along the way, if they are interested or progress to the point where they need a "real" language, they'll have all that knowledge, rather than run into the syntax-error and compiling brick-wall..

Share Improve this answer

Follow

answered Dec 9, 2008 at 10:18

dbr

169k ● 69 ● 283 ● 347

0

It seems that if you are trying to prepare our team for a future in programming that C(++) ma be the better route. The promise of general programming languages that are built with visual building blocks has never seemed to materialize and I am beginning to wonder if they ever will. It seems that while it can be done for specific problem domains, once you get into trying to solve many general problems a text based programming language is hard to beat.

At one time I had sort of bought into the idea of executable UML but it seems that once you get past the object relationships and some of the process flows UML would be a pretty miserable way to build an app. Imagine

trying to wire it all up to a GUI. I wouldn't mind being proven wrong but so far it seems unlikely we'll be point and click programming anytime soon.

Share Improve this answer

Follow

answered Aug 17, 2008 at 21:02

N8g

636 ● 2 ● 8 ● 19

0

I started with LabVIEW about 2 years ago and now use it all the time so may be biased but find it ideal for applications where data acquisition and control are involved.

We use LabVIEW mainly for testing where we take continuous measurements and control gas valves and ATE enclosures. This involves both digital and analogue input and outputs with signal analysis routines and process control all running from a GUI. By breaking down each part into subVIs we are able to reconfigure the tests with the click and drag of the mouse.

Not exactly the same as C/C++ but a similar implementation of measurement, control and analysis using Visual BASIC appears complex and hard to maintain by comparision.

I think the process of programming is more important than the actual coding language and you should follow the style guidelines for a graphical programming language. LabVIEW block diagrams show the flow of data (Dataflow programming) so it should be easy to see

potential race conditions although I've never had any problems. If you have a C codebase then building it into a dll will allow LabVIEW to call it directly.

Share Improve this answer Follow

edited Aug 19, 2008 at 17:57

answered Aug 19, 2008 at 16:31

Swinders

2,281 ● 4 ● 29 ● 43

0

There are definitely merits to both choices; however, since your domain is an educational experience I think a C/C++ solution would most benefit the students. Graphical programming will always be an option but simply does not provide the functionality in an elegant manner that would make it more efficient to use than textual programming for low-level programming. This is not a bad thing - the whole point of abstraction is to allow a new understanding and view of a problem domain. The reason I believe many may be disappointed with graphical programming though is that, for any particular program, the incremental gain in going from programming in C to graphical is not nearly the same as going from assembly to C.

Knowledge of graphical programming would benefit any future programmer for sure. There will probably be opportunities in the future that only require knowledge of graphical programming and perhaps some of your

students could benefit from some early experience with it. On the other hand, a solid foundation in fundamental programming concepts afforded by a textual approach will benefit **all** of your students and surely must be the better answer.

Share Improve this answer Follow

answered Sep 17, 2008 at 7:07

b3.

7,155 ● 2 ● 36 ● 49

0

> The team captain thinks that LabVIEW is better for its ease of learning and teaching. Is that true?

I doubt that would be true for any single language, or paradigm. LabView could surely be easier for people with electronics engineering background; making programs in it is "simply" drawing wires. Then again, such people might already be exposed to programming, as well.

One essential difference - apart from from the graphic - is that LV is demand based (flow) programming. You start from the outcome and tell, what is needed to get to it. Traditional programming tends to be imperative (going the other way round).

Some languages can provide the both. I crafted a multithreading library for Lua recently (Lanes) and it can be used for demand-based programming in an otherwise imperative environment. I know there are successful

robots run mostly in Lua out there (Crazy Ivan at Lua Oh Six).

Share Improve this answer Follow

edited Sep 17, 2008 at 18:36

answered Sep 17, 2008 at 18:04

akauppi
18k ● 15 ● 102 ● 124

0

Have you had a look at the Microsoft Robotics Studio? http://msdn.microsoft.com/en-us/robotics/default.aspx

It allows for visual programming (VPL): http://msdn.microsoft.com/en-us/library/bb483047.aspx as well as modern languages such as C#. I encourage you to at least take a look at the tutorials.

Share Improve this answer Follow

answered Dec 9, 2008 at 9:30

Kirill Osenkov
8,966 ● 2 ● 37 ● 38

0

My gripe against Labview (and Matlab in this respect) is that if you plan on embedding the code in anything other than x86 (and Labview has tools to put Labview VIs on ARMs) then you'll have to throw as much horsepower at the problem as you can because it's inefficient.

Labview is a great prototyping tool: lots of libraries, easy to string together blocks, maybe a little difficult to do

advanced algorithms but there's probably a block for what you want to do. You can get functionality done quickly. But if you think you can take that VI and just put it on a device you're wrong. For instance, if you make an adder block in Labview you have two inputs and one output. What is the memory usage for that? Three variables worth of data? Two? In C or C++ you know, because you can either write `z=x+y` or `x+=y` and you know exactly what your code is doing and what the memory situation is. Memory usage can spike quickly especially because (as others have pointed out) Labview is highly parallel. So be prepared to throw more RAM than you thought at the problem. And more processing power.

In short, Labview is great for rapid prototyping but you lose too much control in other situations. If you're working with large amounts of data or limited memory/processing power then use a text-based programming language so you can control what goes on.

Share Improve this answer

Follow

answered Aug 24, 2009 at 18:36

Stephen Friederichs

1,059 ● 6 ● 12

0

People always compare labview with C++ and day "oh labview is high level and it has so much built in functionality try acquiring data doing a dfft and displaying the data its so easy in labview try it in C++".

Myth 1: It's hard to get anything done with C++ its because its so low level and labview has many things

already implemented. The problem is if you are developing a robotic system in C++ you MUST use libraries like opencv , pcl .. ect and you would be even more smarter if you use a software framework designed for building robotic systems like ROS (robot operating system). Therefore you need to use a full set of tools. Infact there are more high level tools available when you use, ROS + python/c++ with libraries such as opencv and pcl. I have used labview robotics and frankly commonly used algorithms like ICP are not there and its not like you can use other libraries easily now.

Myth2: Is it easier to understand graphical programming languages

It depends on the situation. When you are coding a complicated algorithm the graphical elements will take up valuable screen space and it will be difficult to understand the method. To understand labview code you have to read over an area that is O(n^2) complexity in code you just read top to bottom.

What if you have parallel systems. ROS implements a graph based architecture based on subcriber/publisher messages implemented using callback and its pretty easy to have multiple programs running and communicating. Having individual parallel components separated makes it easier to debug. For instance stepping through parallel labview code is a nightmare because control flow jumps form one place to another. In ROS you don't explicitly 'draw out your archietecture like in labview, however you

can still see it my running the command ros run rqt_graph (which will show all connected nodes)

"The future of programming is graphical." (Think so?)

I hope not, the current implementation of labview does not allow coding using text-based methods and graphical methods. (there is mathscript , however this is incredibly slow)

Its hard to debug because you cant hide the parallelism easily.

Its hard to read labview code because there you have to look over so much area.

Labview is great for data aq and signal processing but not experimental robotics, because most of the high level components like SLAM (simultaneous localisation and mapping), point cloud registration, point cloud processing ect are missing. Even if they do add these components and they are easy to integrate like in ROS, because labview is proprietary and expensive they will never keep up with the open source community.

In summary if labview is the future for mechatronics i am changing my career path to investment banking... If i can't enjoy my work i may as well make some money and retire early...

Share Improve this answer Follow

edited Sep 14, 2013 at 12:42

answered Sep 14, 2013 at 12:35

SentinalBais

35 ● 4