Dependencies and Compilation

Building and Running the Pinball Autopilot on the Raspberry Pi

The source code lies and building happens inside the folder called `pinballProject`. For a fresh build, you need a not too old version of `make`, a C++ compiler (we only tested `g++`), an internet connection (for downloading the Kieler Compiler), the libraries `wiringPi` and `opencv` as well as the command-line tools mega-cmd and `python2`. The tool for mega.nz (and also `python2`) can be omitted (see below on how to compile without them), everything else except for `opencv` comes pre-installed with Raspbian. For `opencv` you need a 4.x version, 4.2.0 ought to work for example. Make sure to enable the option that the entries for `pkg-config` get configured. It can be built from source on the Raspberry Pi in a few hours, make sure not to use too many threads for compilation because of limited RAM, also perhaps increase the swap.

With all the prerequisites met, you can use the `Makefile` to build everything.

Command	Functionality (All the builds are incremental, i.e. only happen once when the source code is unchanged.)
`make`	same as `make build_all`
`make build_all`	Build the main executable as well as the tools `video_capture` and `calibrate`.
`make build`	Build the main executable “`main`”.
`make run`	Build and execute the main executable.
`make ...xyz`	Make file called “`...xyz`”, if a recipe exists. Most notably for `...xyz` = `calibrate`
`make capture`	Build `video_capture` and execute it. This tool can only be executed on the Pi and can record a video during manual gameplay which is then uploaded to mega.nz with a new numerical file-name.
`make get`	Executes a python script that can (interactively) get a recorded video from the mega.nz folder. Intended for getting video input files transferred to a different machine (not the Pi) for testing the image detection.
`make clean`	Removes all generated files except for `_pi_logged_in`, downloaded videos and the downloaded Kieler Compiler.
`make clean_all`	Like `make clean`, but without the exceptions. This will delete everything in the folder `video`!

For instructions on how to use the `./calibrate`, see World Coordinate System.

Working without mega.nz (and python): On the pi, you need to create a file called `_pi_logged_in` (e.g. with `touch _pi_logged_in`) before calling `make` for the first time. On a different machine, you don’t need to do this. You cannot use `make capture` or `make get` without these command-line tools.

If future updates of `mega-cmd` or the command-line Kieler Compiler `kico.jar` include relevant breaking changes, you might need to resolve those problems (work without `mega-cmd` or acquire a `kico.jar` from around March 2020).

Setting up the Pi for Better Real Time Performance

The program will stick its latency-sensitive threads for controlling the IO pins and for image processing to the cores 3 and 4 of the 4-core Raspberry Pi. To ensure best real-time performance, please configure the kernel parameter `isolcpus=2,3` (they count starting from 0) to the file `/boot/cmdline.txt` and reboot. This will make sure that the operating system doesn’t schedule any other tasks on those cores.

Building and Running on Any Other Linux Machine

The program can be built and partially run on other machines. There you potentially need to install all the requirements yourself. However, `wiringPi` is not needed. Also don’t use a machine where the current user is called `pi`, since that would confuse the compilation flags. We’ve observed problems with a too old version for `make`, without knowing the exact lower bound, 4.2.1 is definitely new enough. Possibly `python2` needs to be installed on your machine. Compared to a Raspberry Pi, compilation of `opencv` on your machine is going to be a lot more fun (way faster and no issues with too little RAM). Regarding versions, see the relevant section above; note that `pkg-config` is needed, too, in case you don’t have that installed already.

Running the `make` commands works like on the Pi. Just `make capture` is not available. The main program will behave differently. It will prompt you for the name (without extension) of a video file in the `video` folder. Make sure to get some file in there; at the time of writing this, there is a file called `1` in the mega.nz-folder that you can get via `make get`, as well as a few more named video files that were created before we had the `make capture` command.

After choosing a video file, the program starts using the video input in “real time”; however, since things are not really real-time, the logic controller part does not work too well. (Simulating the flow of time properly is not fundamentally a problem but something we didn’t finish implementing.) This means you should mostly ignore the debug-printouts about ticks happening. What you can really test is the image processing / ball detection. This works better than on the Pi, because we are providing controls for pausing or changing the playback speed. Press a key while any of the windows opened by the main program is in focus.

Key	Playback effect
`<space>`	play/pause

+	increase playback speed ×2
-	decrease playback speed ×0.5
.	while paused, advance by one frame

If you recalibrate the coordinate system on the Pi, you might want to transfer the config file for the coordinate system (however there is nothing really depending on the coordinate transformation being correct inside the image detection anyways).

The ability to compiler the whole project off the Pi allows for catching type errors etc. more quickly, since compilation on the Pi tends to be a lot slower and also there's not the option to use the Kieler IDE on it (so you'd maybe also need to transfer the code). The `pinballProject` folder contains an Eclipse project that can be used with Kieler. Building from the Kieler IDE still happens by invoking the Makefile. There should be a build configuration using `make` included in the project. (I'm only specifying "should" since we didn't really test the experience of importing the project into a workspace too often.)