Beamer Module - Billard@ISEM

Release 0.1

Mathis Wolter

Jan 14, 2026

CONTENTS:

The Beamer Module is a part of the Billard@ISEM system, ultimately aiming at teaching students relevant mechanical design principles as well as provide a research environment for agile product development methods. It provides a visual output that can be perfectly cropped to the billard table using included calibration features.

CHAPTER

ONE

BEAMER MODULE FOR THE BILLARD@ISEM SYSTEM

This is the implementation for a Beamer Module as defined in documentation.

Using a Beamer above the billard table, the system gains the ability to project images directly onto the table. A perspective warp can be applied after manual calibration for the projection to perfectly fit the table. If the system running this software is able to play sounds (like most commercial beamers), this module can also output sound and set the volume.

Run `Beamer.py` with a connected beamer to start the webserver and local GUI/display thread.

1.1 Installation and assumptions

- Hardware recommendations: A beamer connected to a small computer (like a Raspberry PI) running a linux version (like Raspberry PI OS). If the beamer has different resolution than 1080x1920, you will need to individually recreate the configuration image in `static/grid.bmp`.

1. Assumptions about the system you are installing this on: and `git` are available. This is the case on most linux versions, especially Raspberry PI OS (only tried on a non-headless version). For debugging minor knowledge of `systemd/systemctl` and `journalctl` are beneficial.
2. Clone the repository with submodules: `git clone --recurse-submodules https://github.com/ISEM-TUHH/billard-beamer-module.git`
    - For ease of installation, clone it into your home directory (`/home/<username>/`)
3. Go into the newly created directories installation toolbox and run the installation script.
    - Edit `install/billard-beamer-server.service` to have the correct username (replace `beamer-pi`).
    - Edit `config/config.json` to have the correct informations for your setup. If you want to develop, also edit the `config/test_config.json` file.
    - `cd billard-beamer-module/install`
    - `sudo chmod +x install.sh && sudo ./install.sh` (sudo is required as this installs a new `systemctl` service)
        - This install and starts a new systemd service `billard-beamer-server.service`.
4. If successful, a test image should be displayed and the server should be availabe under port 5000 of the given IP address: `http://<address>:5000`
    - This website is very barebones and mostly used for configuration and testing

1.2 Configuration

This part is concerned with setting up the display area to perfectly match the billard table.

1. Go to the modules configuration website under `http://<address>:5000/v1/config`
2. This displays a grid on the beamer.
    - By selecting a corner on the website and clicking somewhere on the image, a marker is displayed at corresponding place on the beamer
    - Repeat until you perfectly match the corner you are wanting to fixate
        - If you are also using the camera module, these should be configured on the same corners: the most outer corners of the camera markers
    - Repeat for all corners
        - If some corners are not in the display area of the beamer, you can manually edit the pixel position of the corners on the website.
    - Click on the "Apply configuration" button to see your changes applied.
        - If you want to iterate, reload the page and repeat. Reloading the page resets the configuration
        - If you want to keep the configuration, click on "Save the configuration". You need to have a configuration applied previously for this to apply.

1.3 Documentation

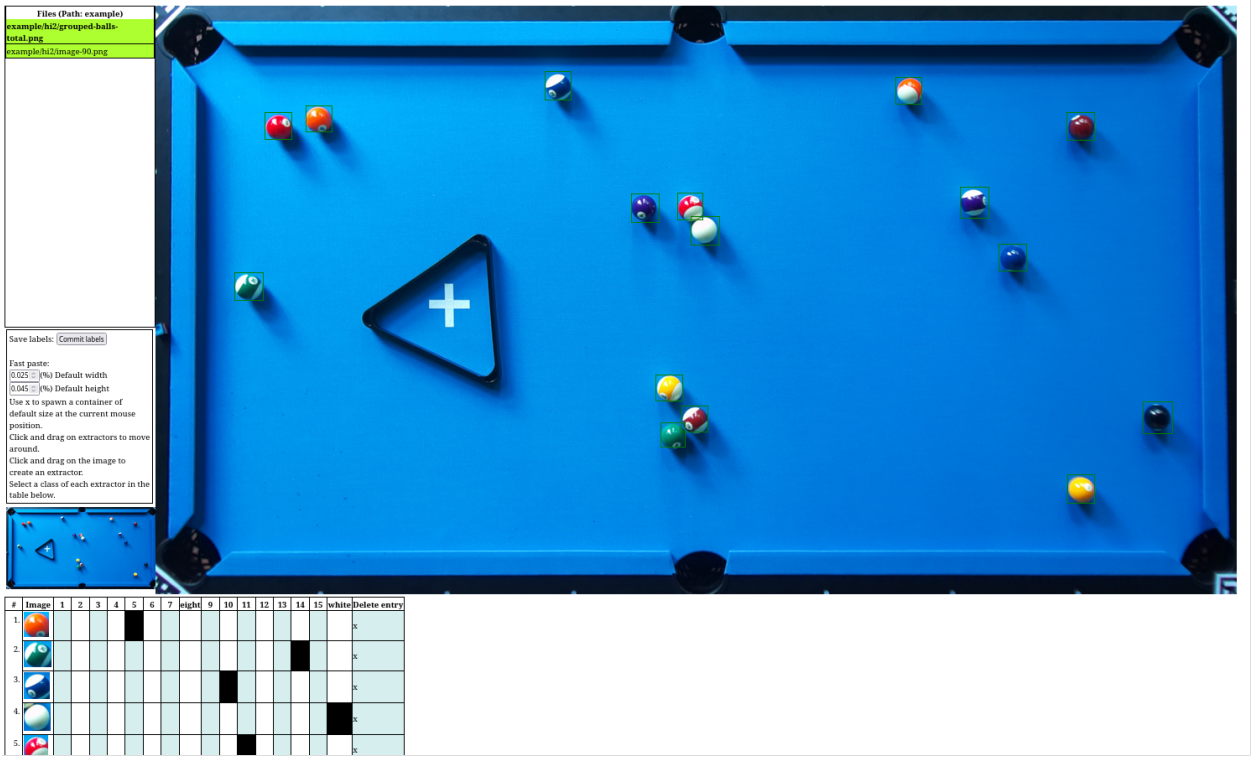
Documentation can be generated using sphinx, which also gets installed at installation time into the virtual environment. To build the html documentation, use

```
source .venv/bin/activate
cd docs
make html
```

Alternatively, you can build a PDF version by using `make latexpdf`. This requires a full LaTeX installation on your computer.

Documentation for the web API is also available on the main website, all available endpoints under `http://<address>:5000/api-doc` and every endpoint has documentation under `http://<address>:5000/<endpoint>.doc`.

Yolo Table Label



CHAPTER

TWO

API

Beamer	The Beamer Module for the Billard@ISEM system.

2.1 Beamer

The Beamer Module for the Billard@ISEM system.

This module supplies the class *Beamer* and the function *display_image*. *Beamer* supplies the webserver and API, while *display_image* hosts a *cv2.imshow* thread, actually showing the image.

Beamer.**update_frame_flag**

if true, the running *display_image* instance will show the currently defined *frame*

Type
bool

Beamer.**frame**

cv2 image that gets shown

Type
np.ndarray

Beamer.**TEST_MODE**

gets set by the running *Beamer* instance. Corresponds to *Module.TEST_MODE*. If False, the image will be displayed in fullscreen mode.

Type
bool

Beamer.**END_DISPLAY**

if True, the running *display_image* instance will terminate.

Type
bool

Functions

display_image()	This function runs in its own thread, running cv2.imshow to display the global variable *frame* (np.ndarray / cv2 image).

2.1.1 Beamer.display_image

Beamer.**display_image**()

This function runs in its own thread, running cv2.imshow to display the global variable *frame* (np.ndarray / cv2 image). As soon as the flag *update_frame_flag* (bool) is raised, it updates the frame, resetting *updating_frame=False*. If the flag *END_DISPLAY* is raised, this function (the *while True* loop) terminates and closes all windows.

In the test mode (flag ´TEST_MODE` in python, externally set with environment variable *PROD_OR_TEST=TEST*), the image is not displayed in fullscreen.

Classes

Beamer([config, template_folder])	Implementation of the Beamer Module for the Billard@ISEM system.

2.1.2 Beamer

class Beamer.**Beamer**(*config='config/config.json'*, *template_folder='templates'*)

Bases: Module

Implementation of the Beamer Module for the Billard@ISEM system.

Running it also requires the *display_image* function two run in a parallel thread.´ to handle actually displaying the image.

current_dir

Absolute path of this file

Type
str

app

Flask app. This needs to run for the module to run (e.g., *beamer.app.run()*)

Type
flask.Flask

passUnitMatrixWarning

if True, you can save a unit matrix as the transformation matrix

Type
bool

M

perspective warp transformation matrix for transforming the image using *cv2*.

Type
np.ndarray 3x3

trace_type

type of trace that gets drawn in a live inference type mode (see Beamer.put_white_points)

Type
str

trace_length

how long the trace history should be (see Beamer.put_white_points)

Type
int

trace_history

the last *Beamer.trace_length* trace lists (see Beamer.put_white_points)

Type
list

available_sounds

list of all .mp3 files in the *sounds* directory (including subdirectories)

Type
list

state

current state of the Beamer Module. Not really used

Type
str

black_image

a 1080x1920 (full HD) black image

Type
np.ndarray

__init__(*config='config/config.json'*, *template_folder='templates'*)

Initialize a Beamer instance.

This collects all sound resources and uses the parent class *Module* to setup the web API.

Parameters

- **config** (*str, optional*) – Relative path to a config (*.json*) file. Defaults to "config/config.json".
- **template_folder** (*str, optional*) – Relative path to a folder to used as templates by flask. Defaults to "templates".

Methods

__init__(*args, **kwargs)	Initialize self.

configure_camera_answer()

NOT WORKING part of trying to calibrate the beamer based on the Camera Module detecting the beamer.

Returns
an image/jpg mimetype response of the transformed image

Return type
Response

`configure_manual_answer()`

Process the answer of the client when the configuration is in manual mode

Returns
html message containing the points and transformation matrix.

Return type
str

`configure_output()`

Start the config process

This starts the routine to align the beamer output to the pool table as it is recognised by the camera. Displays the configuration image and pings the camera module to measure and respond. (Last not yet implemented)

Currently, this just displays a grid on the beamer for manual configuration.

Returns
rendered configure.html template

Return type
Response

`control_image()`

NOT WORKING part of trying to calibrate the beamer based on the Camera Module detecting the beamer.

Returns
an image/jpg mimetype response of the transformed image

Return type
Response

`display_black_image()`

Display a black image on the beamer

Returns
"Displaying black image", flask requires a response

Return type
str

`do_transform(`*getFromFile=True*`)`

Transform the output image based on the transformation matrix *self.M* or read from the file behing *self.transformPath*.

The type of transformation is determined by *'transformation'* in the config file. Setting it to *'OpenCV'* manually transforms the image using the OpenCV (cv2) library before showing it. *'xrandr'* uses the xrandr system to transform the entire desktop output (this is untested and likely not working!).

Parameters
getFromFile (*bool, optional*) – Decide if the current *self.M* should be used or if the matrix should be loaded from *self.transformationPath* file. Defaults to True.

Returns
description of the transformation process.

Return type
str

`force_restart()`

Stop the system by ending both threads. If this is running on a systemd service that restarts a finished service, this restarts the server. Useful for debugging.

Returns
"Restarting", because flask requires a response

Return type
str

`index()`

Renders the index website.

Returns
rendered index.html page

Return type
Response

`play_sound()`

Plays a sound using mpg123

From a flask request.json, search for the sound behind the json key *sound* in *self.available_sounds* and plays it using mpg123.

Known errors: when using a Raspberry PI 5 connected via HDMI to a beamer and turing the beamer on after the PI, there sometimes is no sound.

Returns
"Playing [file name]" if everythin is fine or a message and 404 when the sound was not found.

Return type
str, (int)

`put_white_points()`

Receive moving balls (with a flask POST request) as json data in the format {"points": [{"x": 123, "y": 345}, ...]} and place them on the canvas.

Coordinates are individually transformed, improving performance compared to drawing them and then transforming the entire image. This leads to small projection errors when drawing stuff around transformed coordinates. Stores the last *self.trace_length* (transformed) coordinates in *self.trace_history* (queue like behaviour).

The type of visual affect must be added via code and could be changed by manipulating *self.trace_type*. Current "doppler" effect displays squares increasing in size and decreasing in brightness by the "age" of the coordinates, simulating the physical doppler effect.

Returns
confirmation "Displaying moving balls"

Return type
str

Todo

- Improve determining the size of the ball
- Add trail of previous balls?
- Add more effects?

`receive_image()`

This API endpoint receives images from the web (game module), transforms and shows it.

POST an image (as the request.data) enocded to uint8.

Returns
"Image received", flask requires response

Return type
str

`sound_volume()`

Change the volume level for sound replay using amixer

From a POST request, use the json data *{"level": 100}* to set the volume between 0 and 100 using the *amixer* utility. The key *level* not in the json data or the level being outside of 0 <= level <= 100 returns in a 403 error.

Returns
"" and the http status code 200 if everything worked or 403 if there is something wrong with the request.

Return type
str, int

`state()`

Returns the current state of the beamer object.

Used to track the current task globally. Not really useful.

Returns
flask.jsonify output of *{'state': self.state}*

Return type
Respone

`update_config_image()`

In the manual config process, this puts live updates from the website (passed as "corner" key in request.json) on the projected image. This allows for the user doing the calibration to see where the corner they selected is.

Returns
"Top", flask requires a response

Return type
str

`update_frame(new_frame, do_transformation=True)`

Pass a new frame to be displayed in the display thread.

From the Beamer object, this updates the global variables *frame*, *update_frame* and *transformed_frame*. The first two signal the *display_image* function running in a different thread to update the displayed image.

If the new frame should be perspectively transformed before displaying, supply *do_transformation=True*. This also saves the last transformed image in the global variable *transformed_frame*.

Parameters

- **new_frame** (*np.ndarray*) – new cv2 image to get displayed
- **do_transformation** (*bool, optional*) – Decide if the image should be transformed before displaying. Only effective if *'transformation': 'OpenCV'* in the used config file. Defaults to True.

CHAPTER

THREE

TO DO

Todo

- Improve determining the size of the ball
- Add trail of previous balls?
- Add more effects?

(The original entry is located in /home/ma4096/Documents/TUHH_Cloud/BA/software/modules/billard-beamer-module/Beamer.py:docstring of Beamer.Beamer.put_white_points, line 11.)

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

I

M

P

R

S

T

U