Iron Car Final Presentation

Simon Rummert - 27.06.2017

What do we want to talk about?

- Goals
- Progress
- Results
- Problems
- Future Work

Goals

- Refactor and update existing Stereo-Vision code
- Improve Stereo Vision Quality
- Implement Collision Avoidance Algorithm

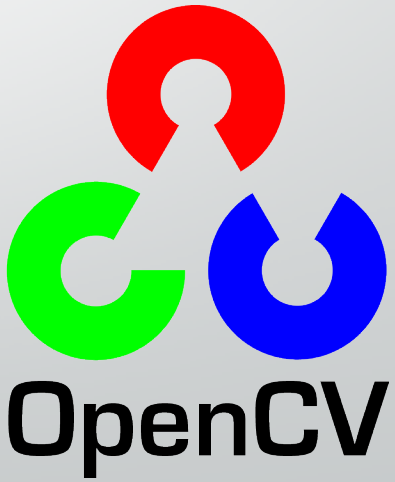
Goals

Refactor and update existing Stereo-Vision code

But Why?

OpenCV 3.0.0 Release Date: June 2015

Improvements over OpenCV 2.4.X:
- Faster
- More stable
- Improved usage of multiprocessor and GPU architectures



Goals

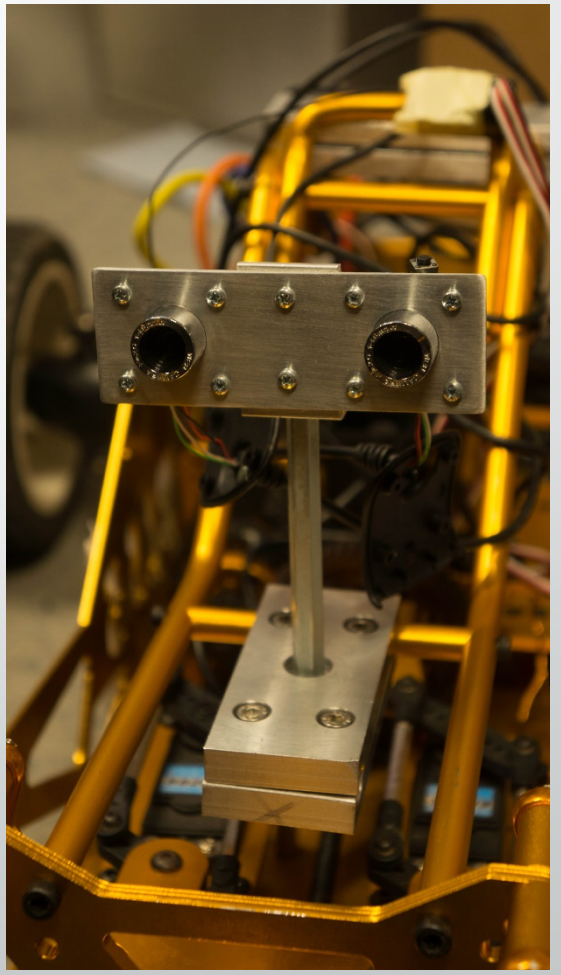
Goals

Improve Stereo Vision Quality

But Why?

Old cameras supply mediocre depth perception because:

- Unsynchronized image capturing
- Random frame drops on change of brightness
- Only support 320x240 resolution



Goals

Improve Stereo Vision Quality

But Why?

We want good results even from simple Block Matching Algorithm
(It is not so accurate, but **faster**)

Goals

Improve Stereo Vision Quality

But Why?

We **DO NOT** want unsatisfying or noisy results

Goals

Goals

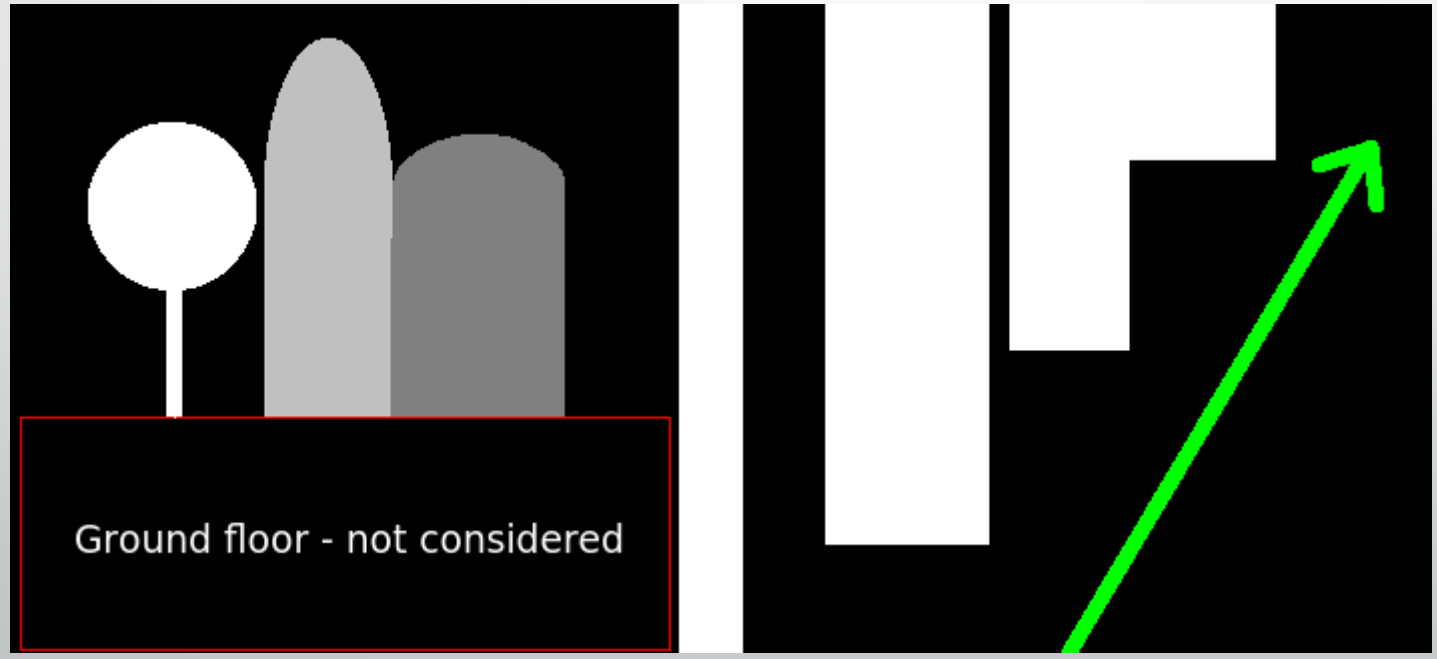
Implement Collision Avoidance Algorithm

Goals

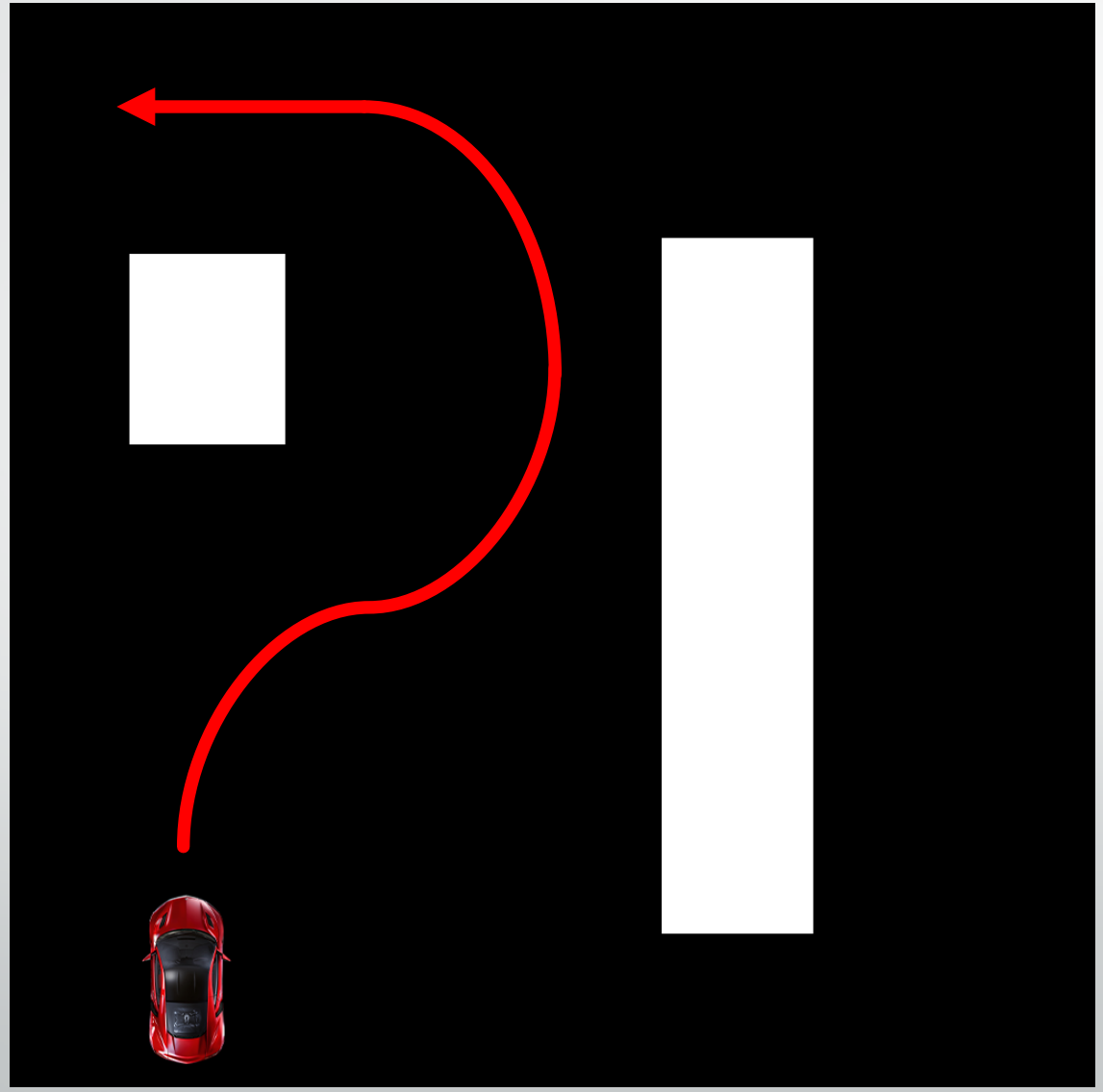
Implement Collision Avoidance Algorithm

But Why?

To avoid collisions...



Goals



Progress

Progress

Refactoring

Update to OpenCV 3.0.0:

- Relatively straight forward
- Mainly changes in the cmake process
- Some function parameters had to be adjusted

Progress

Progress

Stereo Vision Improvements

Used new cameras:

Two PS3 Eye Cameras

- Cheap
- Constant frame rate
- 640x480 at 30 FPS
- **Can be hardware synced**

Progress

Stereo Vision Improvements

What is hardware sync?

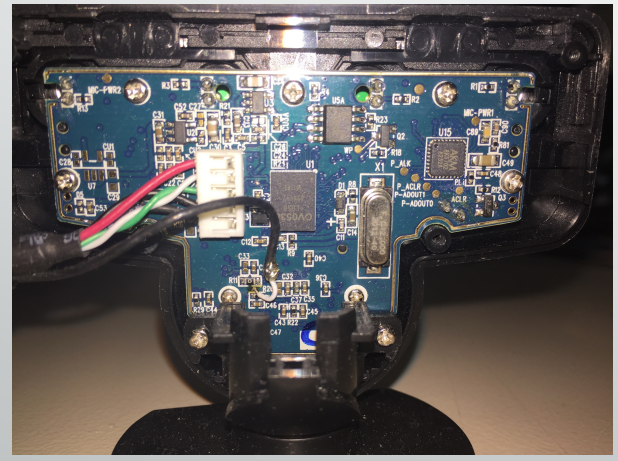
- Two cameras shoot images at exactly the same time
- Resulting image pair produces better depth perception

Progress

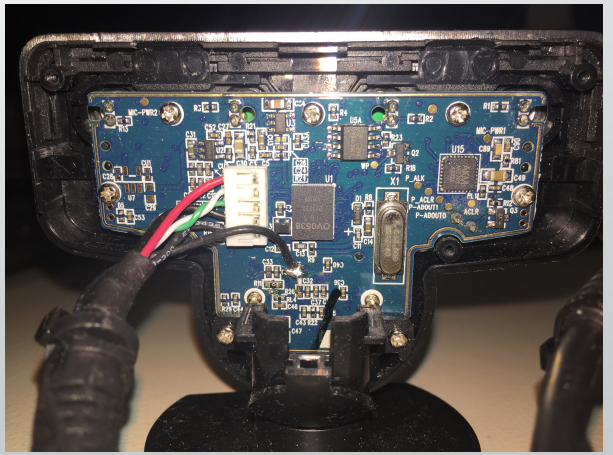
Stereo Vision Improvements

How does it work?

Left camera:
Cable soldered to **VSYNC** pin



Right camera:
Cable soldered to **FSIN** pin

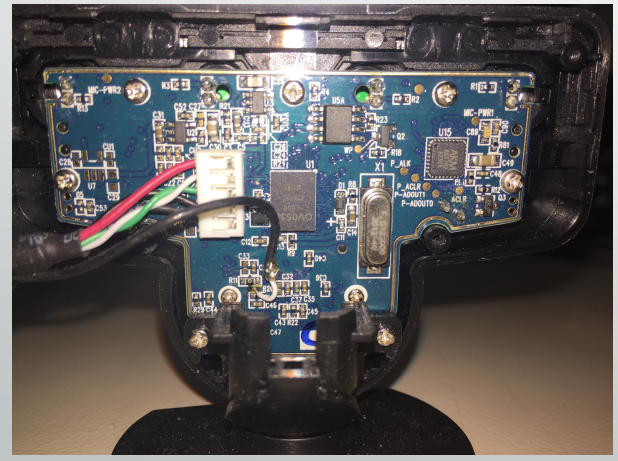


Progress

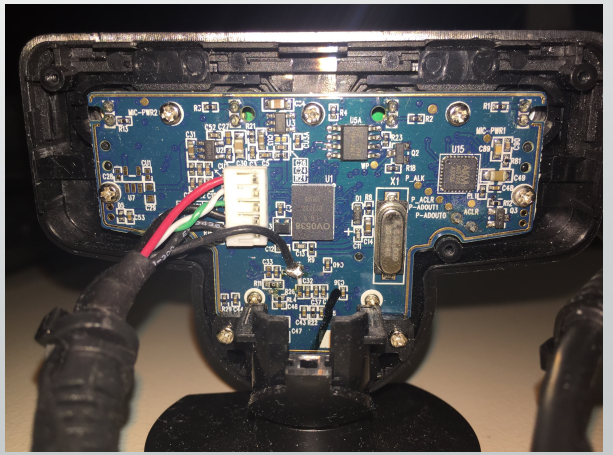
Stereo Vision Improvements

How does it work?

VSYNC
sends signal when image is taken



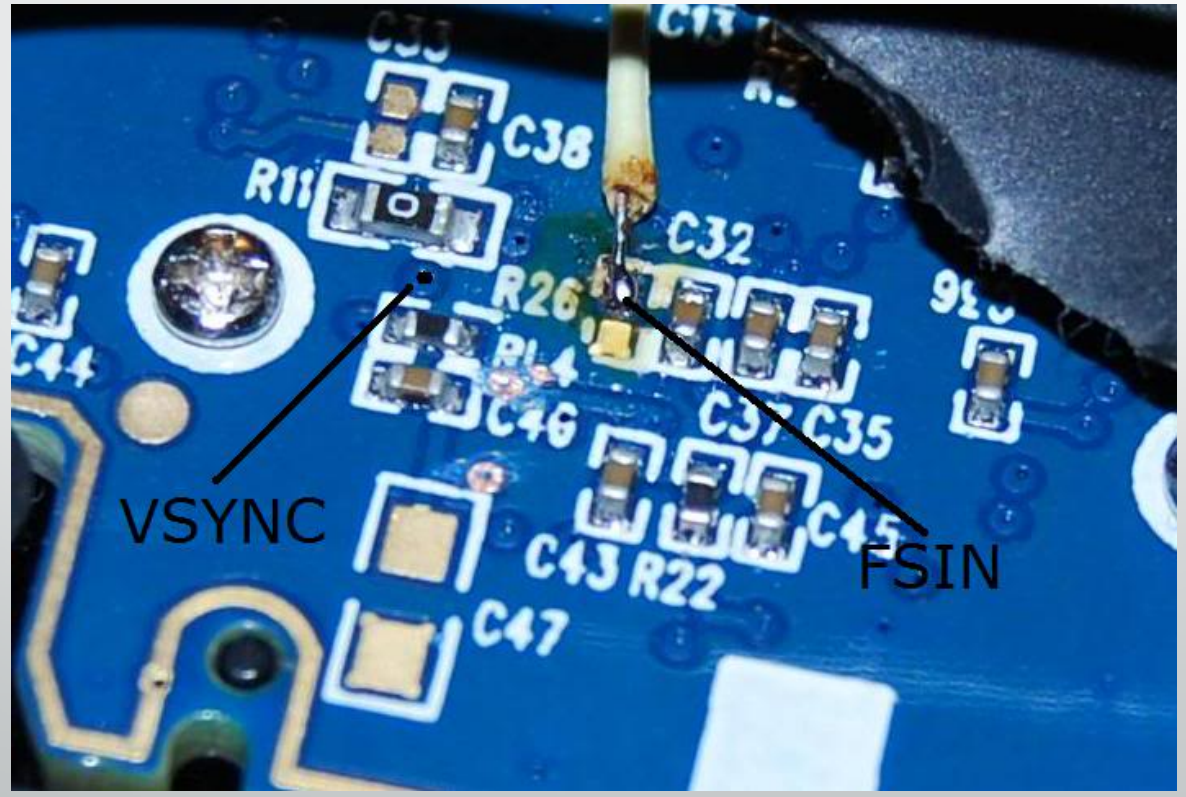
FSIN
receives signal and forces camera to take image



Progress

Stereo Vision Improvements

Pin locations on the board



Progress

Stereo Vision Improvements

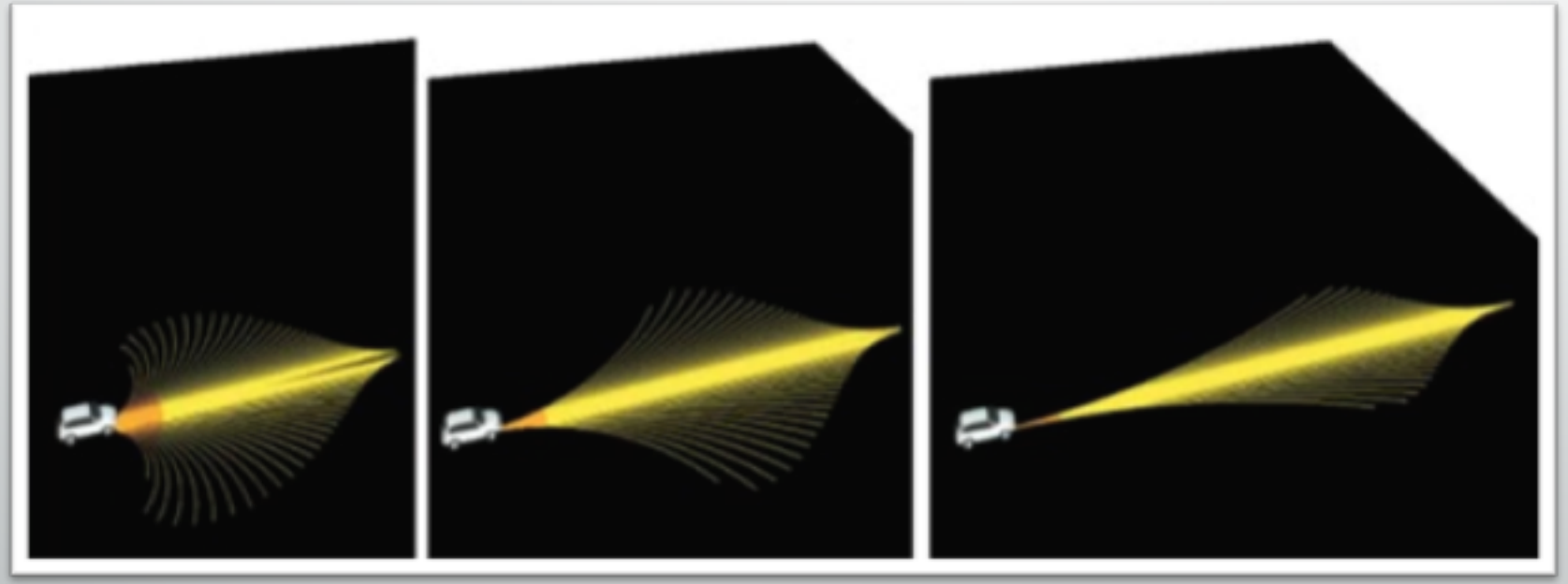
Next step: Building a Mount

Progress

Progress

Collision Avoidance

Idea: Use Tentacles



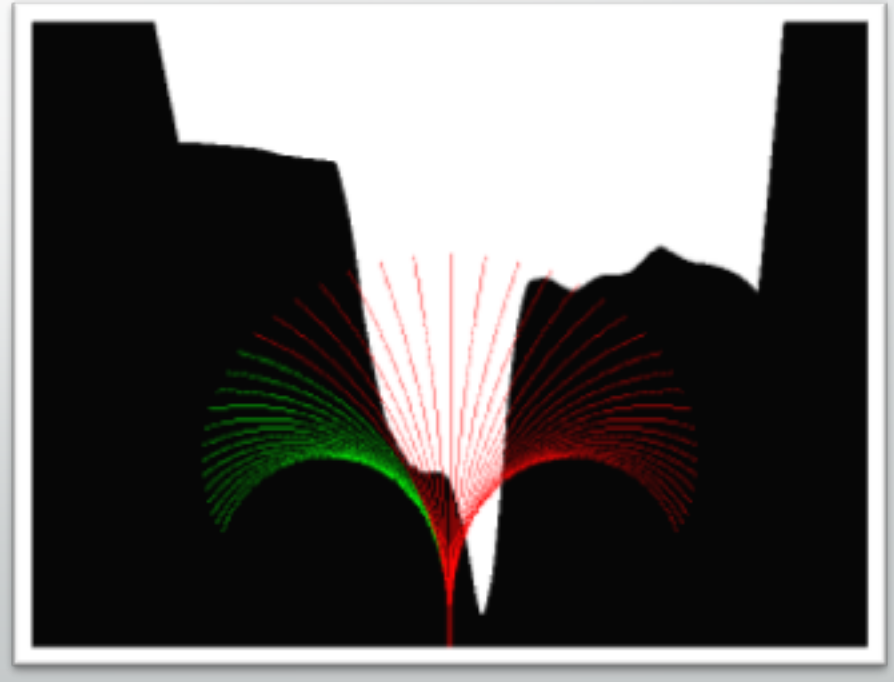
Progress

Collision Avoidance

Idea: Use Tentacles

Top View

With Tentacles



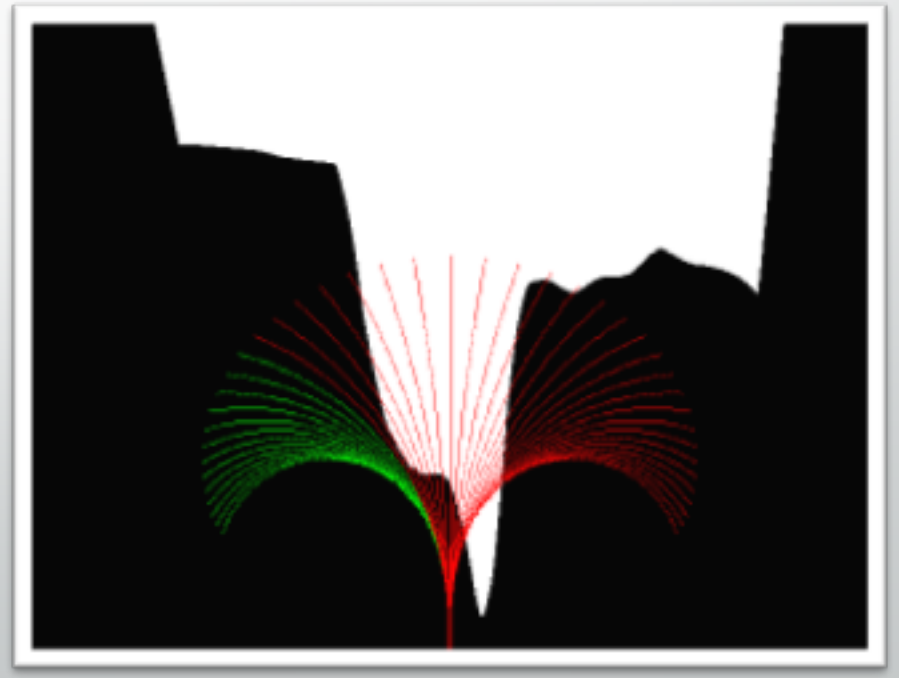
Progress

Collision Avoidance

Idea: Use Tentacles

- One Tentacle represents one steering angle
- **Red** means collision imminent
- **Green** means safe
- Therefore: always choose **green** angle

With Tentacles

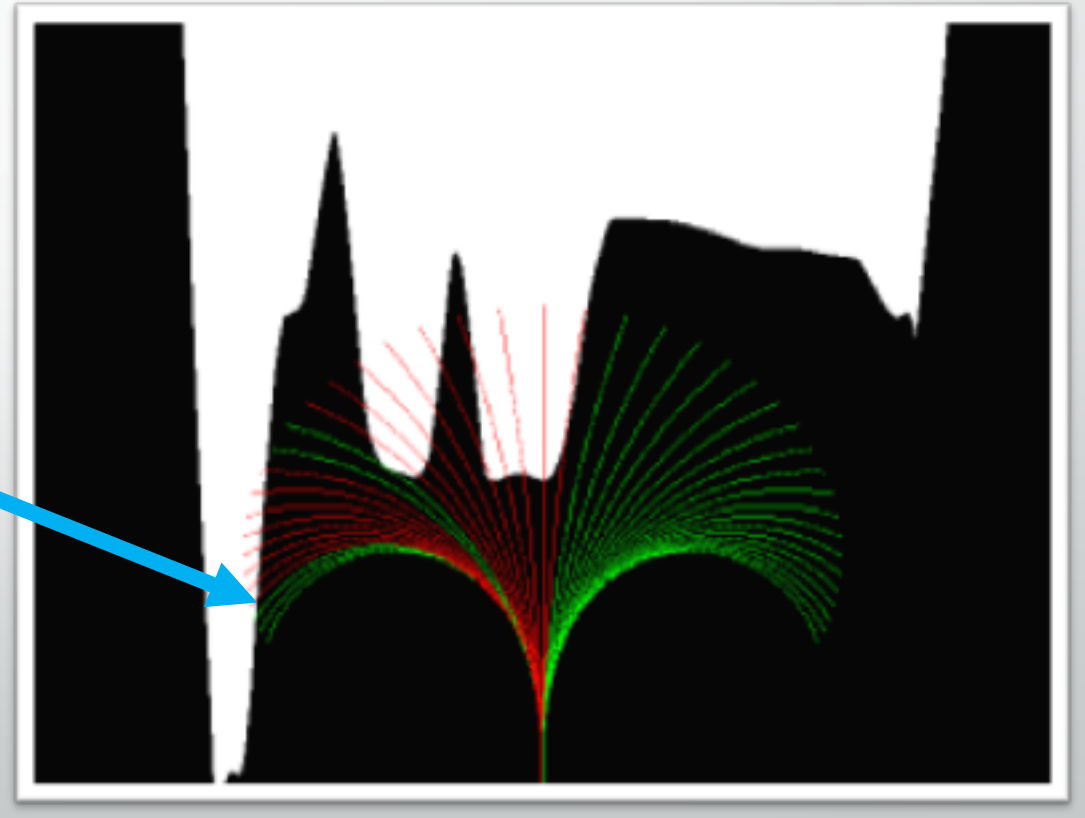


Progress

Collision Avoidance

But which safe angle should I choose?

Should I choose left?

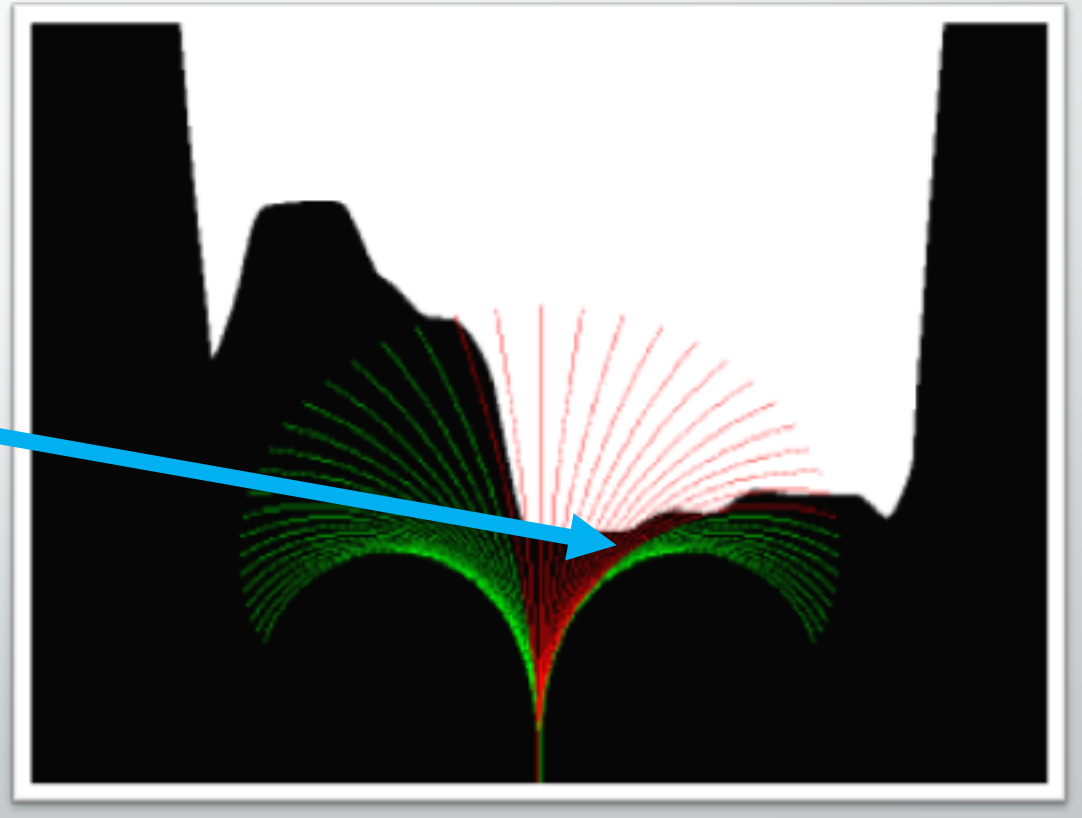


Progress

Collision Avoidance

But which safe angle should I choose?

Should I choose right?

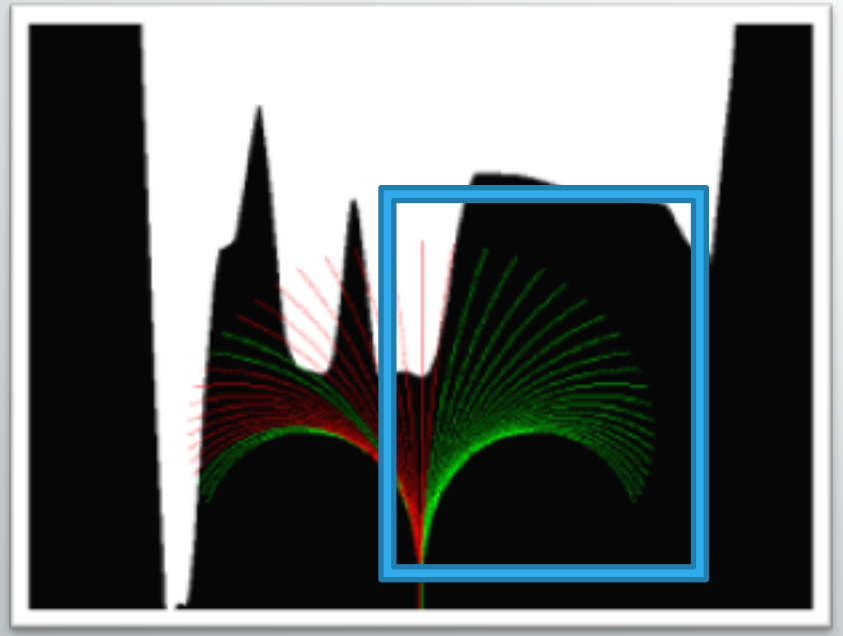


Progress

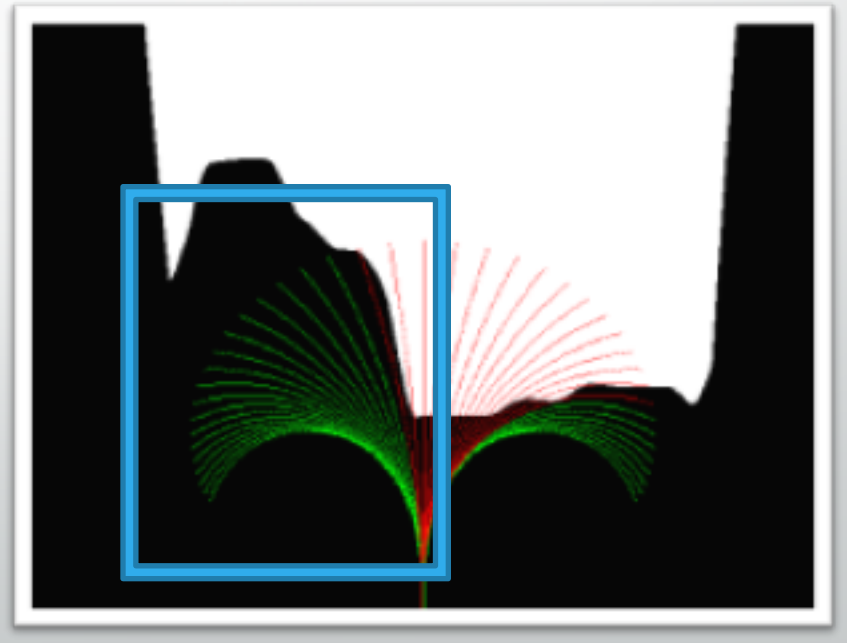
Collision Avoidance

Solution: Assume collision object at the side of most intersecting tentacles

Prefering **RIGHT** side



Prefering **LEFT** side



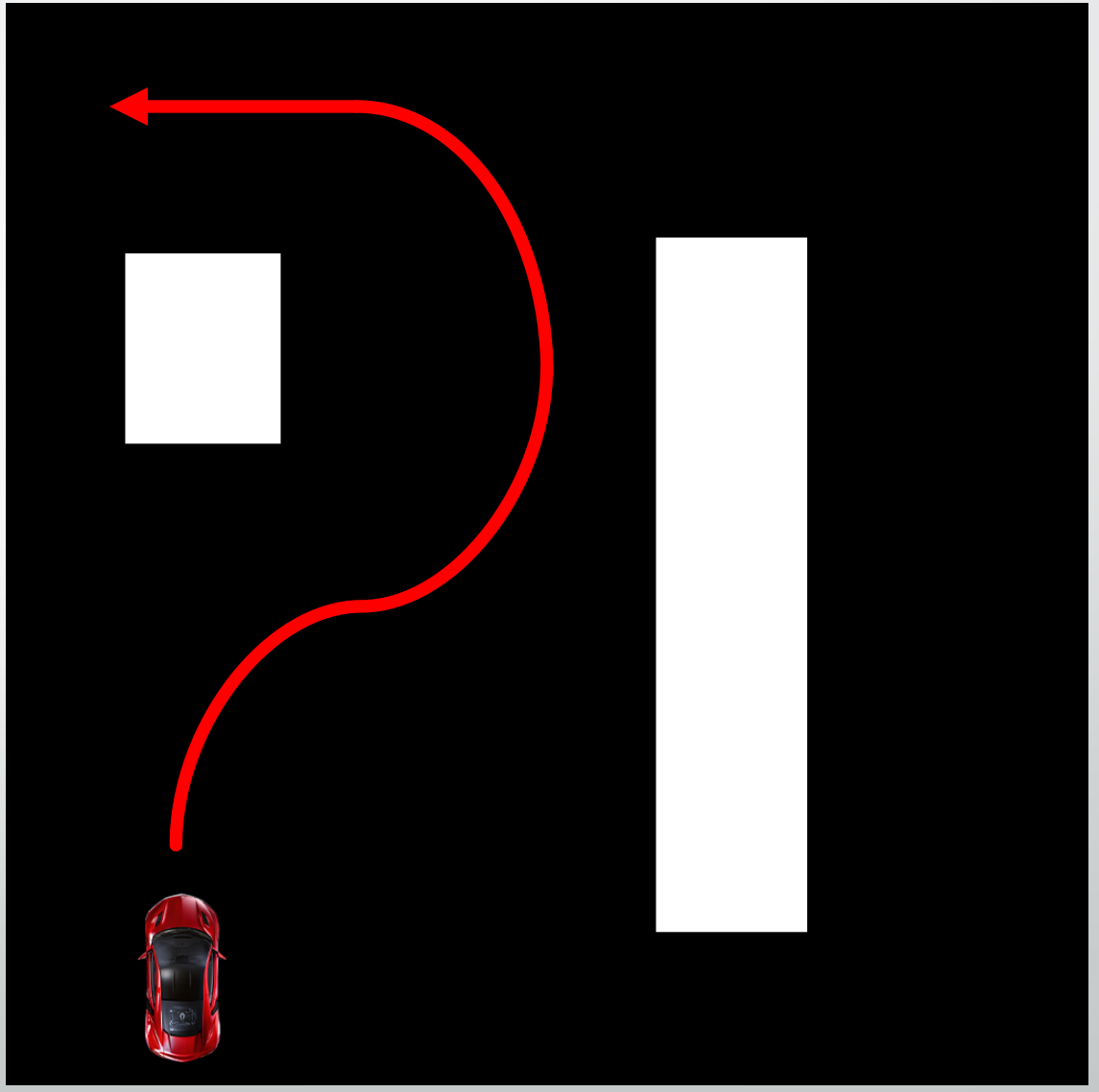
Results

Results

Remember the goal:

Drive around the obstacles

without collision



Results

Results

Why did the car take so long to steer?

Problems

Problems

Measurements have shown:

- Depth map creation with new cameras: **~500ms per iteration**
- Depth map creation with old cameras: **~130ms per iteration**

Why?

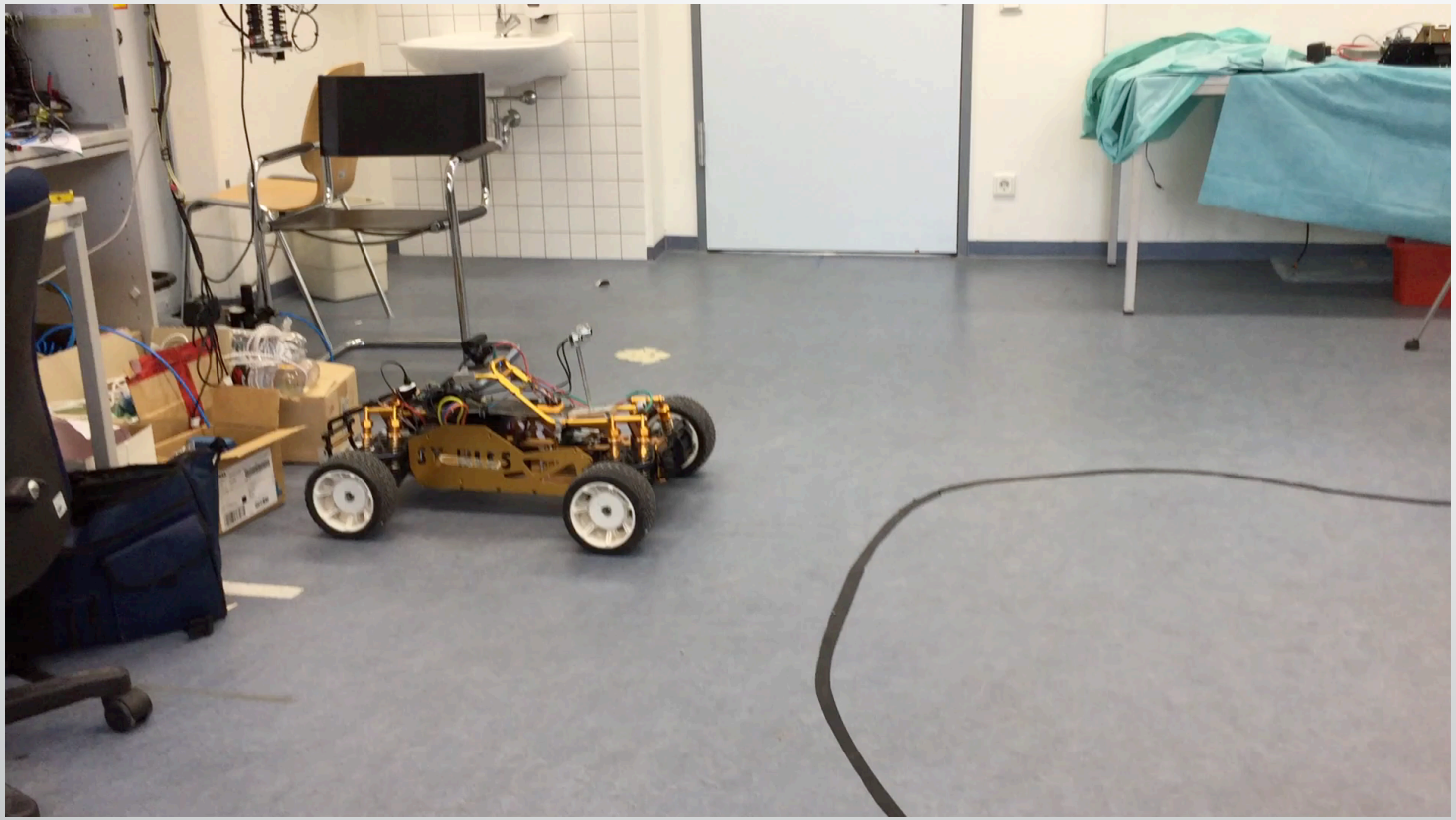
- New cameras have **640x480** resolution
- That is **2 times** the resolution of the old cameras
- In total: **4 times** more pixels for the depth map calculation

Problems

Then why bother at all? Just use the old cameras!

Problems

Results on the same course with old cameras:



Problems

Then why bother at all? Just use the old cameras!

- Remember: We are using the faster, but more **inaccurate** Block Matching algorithm for our depth map

- This means:
  - depth map can get too noisy

- Therefore:
  - Turn down the sensitivity to reduce false positives

Problems

Then why bother at all? Just use the old cameras!

Depth map with old cameras

Depth map with new cameras

Problems

Conclusion

New cameras:

Pro:

- More accurate depth map

Con:

- Slower calculation time

Old cameras:

Pro:

- Overall faster depth map creation

Con:

- Too inaccurate

Future Work

Future Work

- **Use a Multiprocessing/GPU approach for improving the execution time**
  - Use multiple threads for Image Processing
  - Intel's TBB library for multiprocessing is used by OpenCV, with workarounds also compilable on Raspberry Pi 2 and 3
  - Wait for OpenCL to be implemented for Raspberry

THE END