Lego Car

Final Presentation

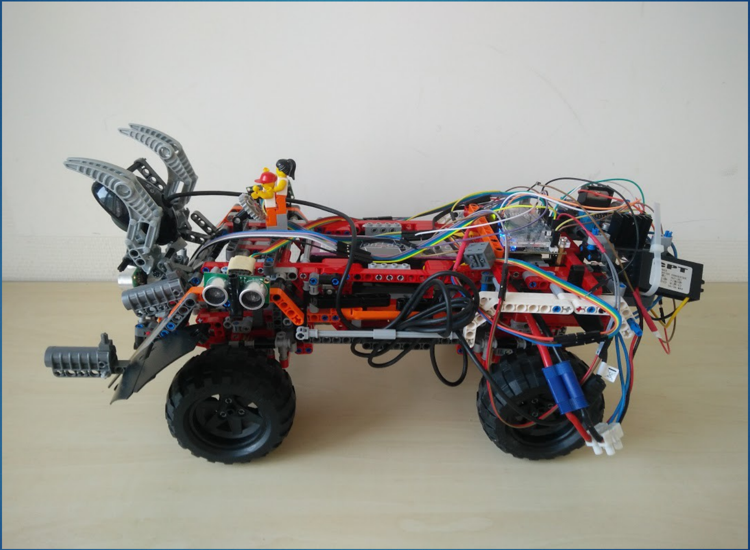
Moritz Dötterl, Berkay Sümer, Abdallah Emad, Heiko Lengenfelder

27.6.2016

Agenda

- Hardware & Wiring
- Adaptive Cruse Control
- Steering Control
- Optimizations

Hardware & Wiring



Hardware

Components

Controll
- DE0-Nano Board
- Raspberry Pi 3

Actuators
- Steering Servo Motor
- Drive Motors

Sensors
- KS103 Ultrasound Sensor
- Camera

Power Supply
- Battery
- H-Bridge
- CPT 15W DC/DC Converter

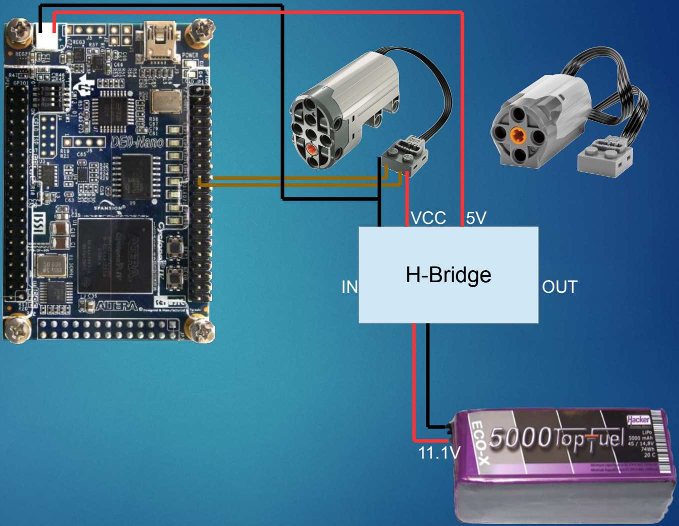
Wiring

Wiring

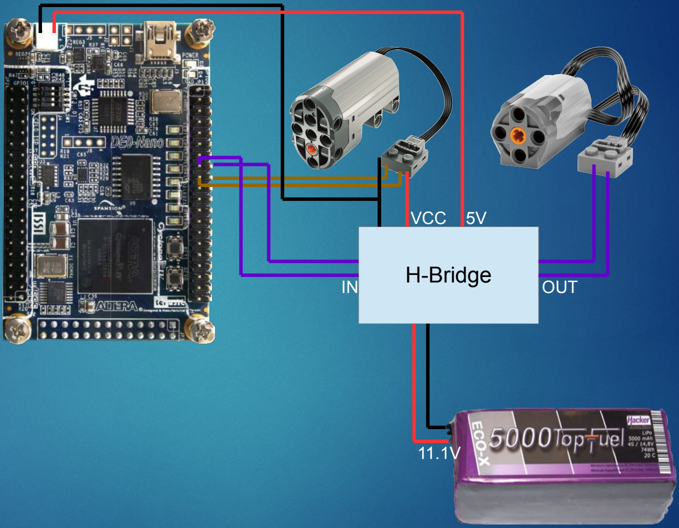
VCC 5V

IN H-Bridge OUT

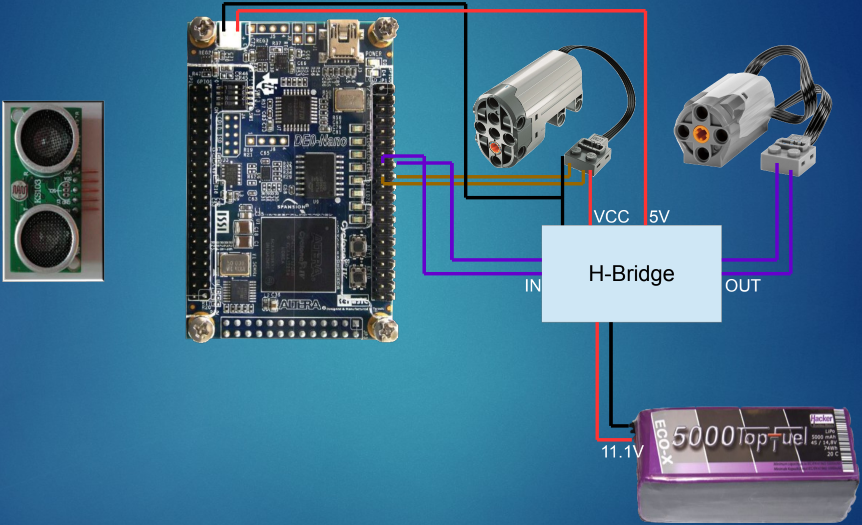
Wiring



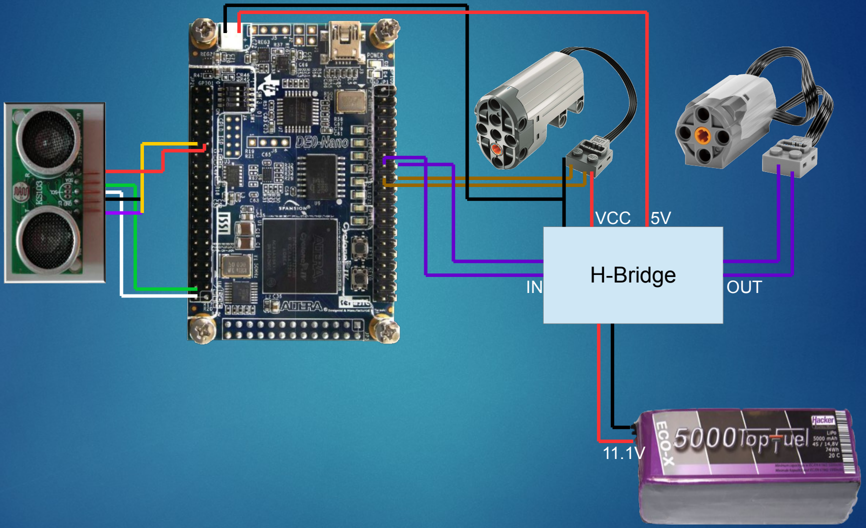
Wiring



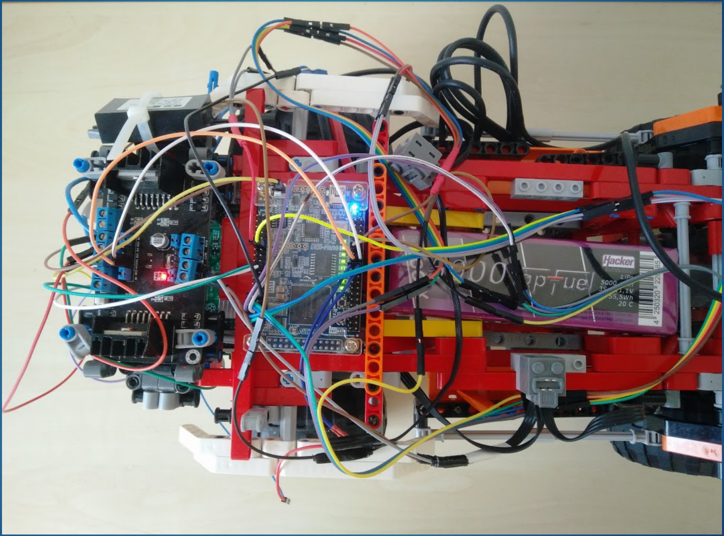
Wiring


DE0-Nano
ALTERA
VCC
5V
H-Bridge
IN
OUT
11.1V
ECO-X
5000 TopFuel
Hacker

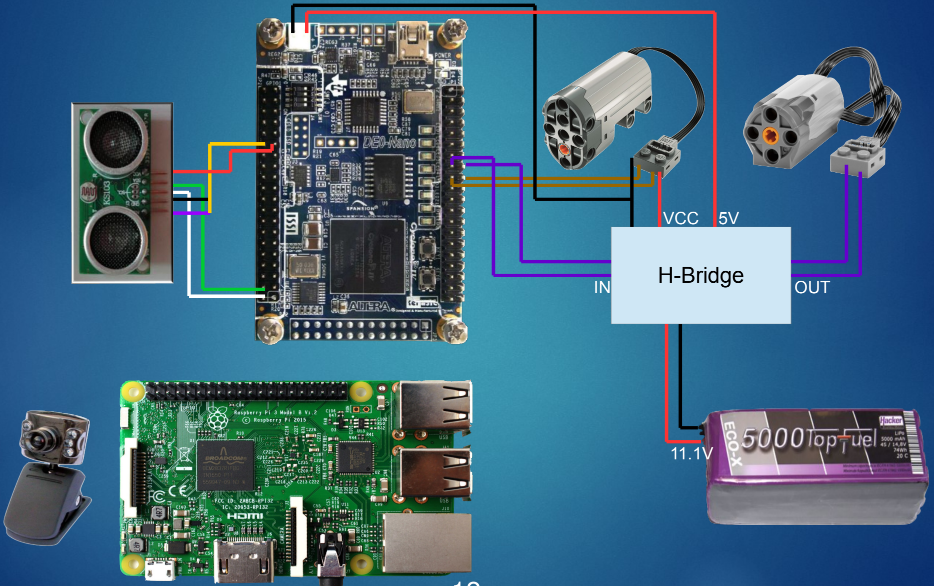
Wiring



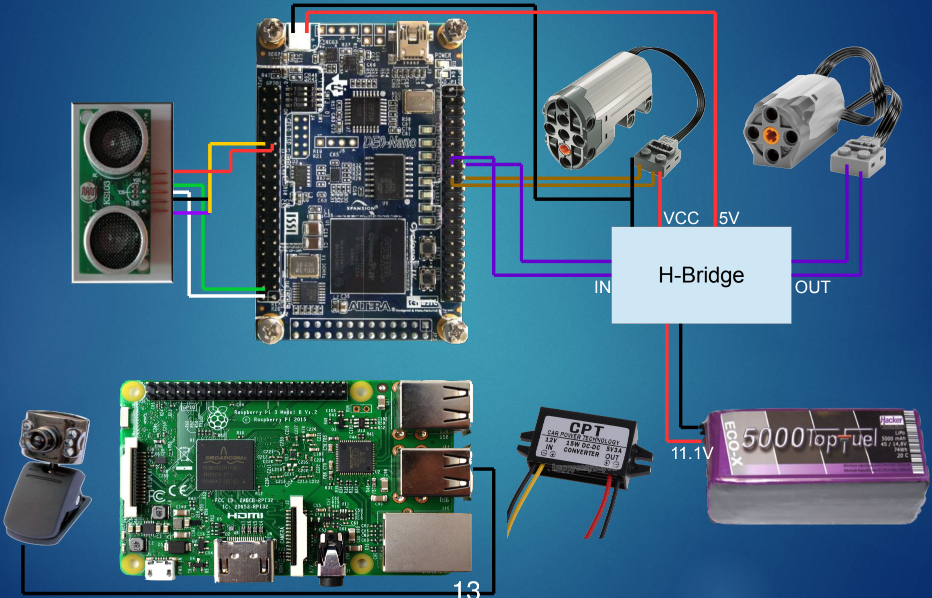
Wiring



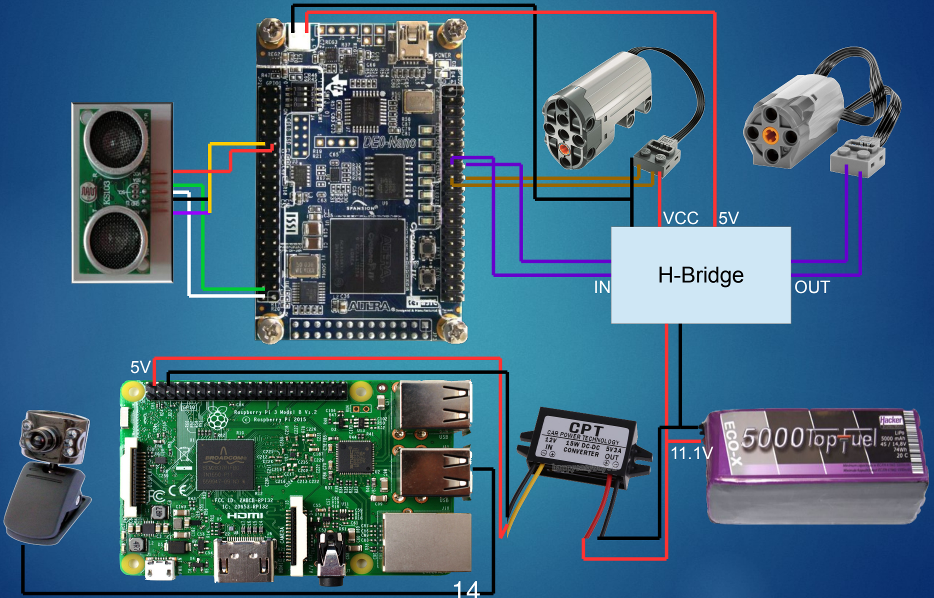
Wiring



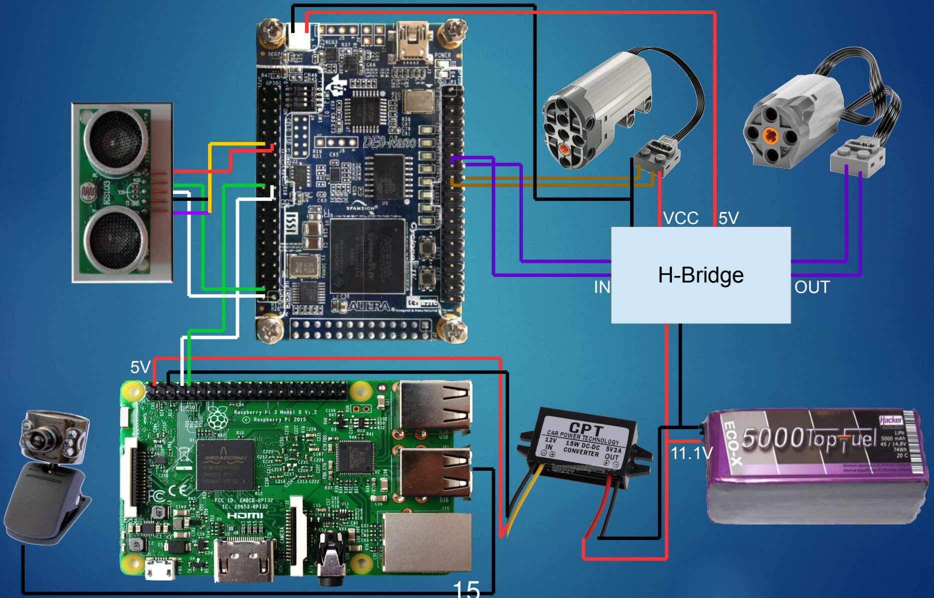
Wiring



Wiring



Wiring



Wiring

Main Functions

Main Functions

- Function: Speed control of the car

Main Functions

- Function: Speed control of the car
  - Input: distance from objects ahead from the ultrasound sensor

Main Functions

- Function: Speed control of the car
  - Input: distance from objects ahead from the ultrasound sensor
  - Communication protocol: Uart communication

Main Functions

- Function: Speed control of the car
  - Input: distance from objects ahead from the ultrasound sensor
  - Communication protocol: Uart communication
  - Output: Duty cycle for the motor

Main Functions

- Function: Speed control of the car
  - Input: distance from objects ahead from the ultrasound sensor
  - Communication protocol: Uart communication
  - Output: Duty cycle for the motor

Note: Time delay after each iteration to prevent ultrasound sensors from crashing

Main Functions

- Function: Speed control of the car
  - Input: distance from objects ahead from the ultrasound sensor
  - Communication protocol: Uart communication
  - Output: Duty cycle for the motor

Note: Time delay after each iteration to prevent ultrasound sensors from crashing

- Function: Car steering to follow the line

Main Functions

- Function: Speed control of the car
  - Input: distance from objects ahead from the ultrasound sensor
  - Communication protocol: Uart communication
  - Output: Duty cycle for the motor

Note: Time delay after each iteration to prevent ultrasound sensors from crashing

- Function: Car steering to follow the line
  - Input: camere input (using Raspberry Pi)

Main Functions

- Function: Speed control of the car
  - Input: distance from objects ahead from the ultrasound sensor
  - Communication protocol: Uart communication
  - Output: Duty cycle for the motor

Note: Time delay after each iteration to prevent ultrasound sensors from crashing

- Function: Car steering to follow the line
  - Input: camere input (using Raspberry Pi)
  - Communication protocol: Uart communication

Main Functions

- Function: Speed control of the car
  - Input: distance from objects ahead from the ultrasound sensor
  - Communication protocol: Uart communication
  - Output: Duty cycle for the motor

Note: Time delay after each iteration to prevent ultrasound sensors from crashing

- Function: Car steering to follow the line
  - Input: camere input (using Raspberry Pi)
  - Communication protocol: Uart communication
  - Output: Duty cycle for the servomotor

Main Functions

- Function: Speed control of the car
  - Input: distance from objects ahead from the ultrasound sensor
  - Communication protocol: Uart communication
  - Output: Duty cycle for the motor

Note: Time delay after each iteration to prevent ultrasound sensors from crashing

- Function: Car steering to follow the line
  - Input: camere input (using Raspberry Pi)
  - Communication protocol: Uart communication
  - Output: Duty cycle for the servomotor

Note: Disable bluetooth on Raspberry Pi 3 in order for Uart communication to work

Adaptive Cruise Control

What is Adaptive Cruise Control?

Adaptive Cruise Control

What is Adaptive Cruise Control?

- Automotive safety feature that allows the vehicle to adapt its speed to the traffic environment

Adaptive Cruise Control

What is Adaptive Cruise Control?

- Automotive safety feature that allows the vehicle to adapt its speed to the traffic environment
- In real cars, a long range radar sensor is used to detect the cars, here we use an ultrasound sensor

Adaptive Cruise Control

What is Adaptive Cruise Control?

- Automotive safety feature that allows the vehicle to adapt its speed to the traffic environment
- In real cars, a long range radar sensor is used to detect the cars, here we use an ultrasound sensor

ACC Algorithm

Adaptive Cruise Control

What is Adaptive Cruise Control?

- Automotive safety feature that allows the vehicle to adapt its speed to the traffic environment
- In real cars, a long range radar sensor is used to detect the cars, here we use an ultrasound sensor

ACC Algorithm

- if distance from ultrasound sensor $\leq$ a minimum set distance ($distance_{min}$) –> stop

Adaptive Cruise Control

What is Adaptive Cruise Control?

- Automotive safety feature that allows the vehicle to adapt its speed to the traffic environment
- In real cars, a long range radar sensor is used to detect the cars, here we use an ultrasound sensor

ACC Algorithm

- if distance from ultrasound sensor $\leq$ a minimum set distance ($distance_{min}$) –> stop
- if distance from ultrasound sensor $\geq$ a maximum set distance ($distance_{max}$) –> drive with maximum speed

Adaptive Cruise Control

What is Adaptive Cruise Control?

- Automotive safety feature that allows the vehicle to adapt its speed to the traffic environment
- In real cars, a long range radar sensor is used to detect the cars, here we use an ultrasound sensor

ACC Algorithm

- if distance from ultrasound sensor $\leq$ a minimum set distance ($distance_{min}$) –> stop
- if distance from ultrasound sensor $\geq$ a maximum set distance ($distance_{max}$) –> drive with maximum speed
- else (distance lies within $distance_{min}$ and $distance_{max}$) –> adapt speed according to distance:

Adaptive Cruise Control

What is Adaptive Cruise Control?

- Automotive safety feature that allows the vehicle to adapt its speed to the traffic environment
- In real cars, a long range radar sensor is used to detect the cars, here we use an ultrasound sensor

ACC Algorithm

- if distance from ultrasound sensor $\leq$ a minimum set distance ($distance_{min}$) –> stop
- if distance from ultrasound sensor $\geq$ a maximum set distance ($distance_{max}$) –> drive with maximum speed
- else (distance lies within $distance_{min}$ and $distance_{max}$) –> adapt speed according to distance:

$$CurrentSpeed = MaximumSpeed * \frac{distance}{distance_{max}}$$

Main Code

The main code runs as follows:

1. Receive distance from ultrasound sensor

Main Code

The main code runs as follows:

1. Receive distance from ultrasound sensor
2. Receive driving angle from Raspberry Pi

Main Code

The main code runs as follows:

1. Receive distance from ultrasound sensor
2. Receive driving angle from Raspberry Pi
3. Calculate driving speed according to the adaptive cruise control algorithm

Main Code

The main code runs as follows:

1. Receive distance from ultrasound sensor
2. Receive driving angle from Raspberry Pi
3. Calculate driving speed according to the adaptive cruise control algorithm
4. Calculate the servomotor's duty cycle to follow the line

Main Code

The main code runs as follows:

1. Receive distance from ultrasound sensor
2. Receive driving angle from Raspberry Pi
3. Calculate driving speed according to the adaptive cruise control algorithm
4. Calculate the servomotor’s duty cycle to follow the line
5. Time delay to prevent ultrasound sensor from crashing

Main Code

The main code runs as follows:

1. Receive distance from ultrasound sensor
2. Receive driving angle from Raspberry Pi
3. Calculate driving speed according to the adaptive cruise control algorithm
4. Calculate the servomotor’s duty cycle to follow the line
5. Time delay to prevent ultrasound sensor from crashing
6. Repeat ..

Steering Control

Approach

- Approximate line
- Calculate direction
- Send data to Nano-Board

Steering Control

Approach

- Approximate line
- Calculate direction
- Send data to Nano-Board

Assumptions

- Vertical line
- Car position
- Line is highest contrast on image

Steering Control

Approach

- Approximate line
- Calculate direction
- Send data to Nano-Board

Assumptions

- Vertical line
- Car position
- Line is highest contrast on image

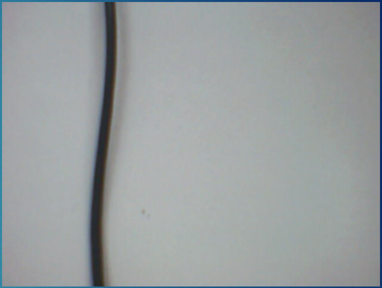
Tools

- Code in C++
- OpenCV

Steering Control

Algorithm

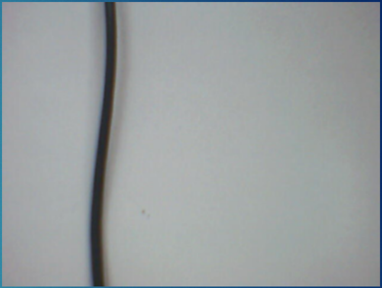
1. Get frame of camera (15 fps)



Steering Control

Algorithm

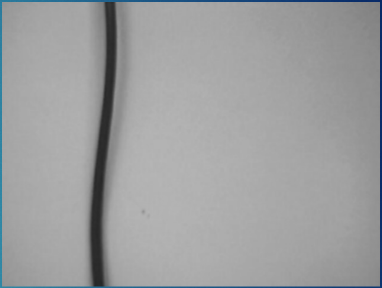
1. Get frame of camera (15 fps)
2. Blur Image
   - Could be done by Gaussian-Filter
   - Do it by hardware (faster)



Steering Control

Algorithm

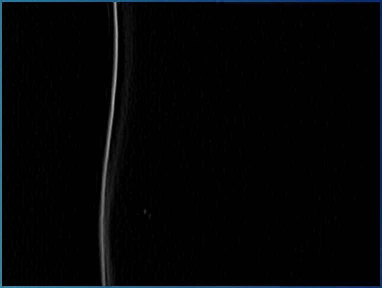
1. Get frame of camera (15 fps)
2. Blur Image
   - Could be done by Gaussian-Filter
   - Do it by hardware (faster)
3. Convert Image from RGB to gray



Steering Control

Algorithm

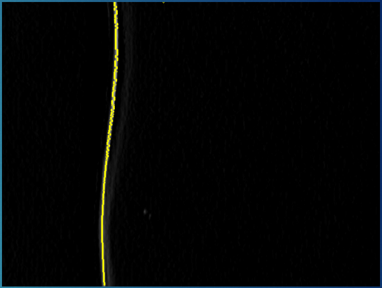
1. Get frame of camera (15 fps)
2. Blur Image
   - Could be done by Gaussian-Filter
   - Do it by hardware (faster)
3. Convert Image from RGB to gray
4. Approximate gradient
   - Only horizontal change
   - Apply Mask $M = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$



Steering Control

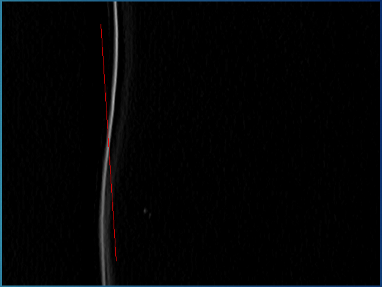
Algorithm

1. Get frame of camera (15 fps)
2. Blur Image
   - Could be done by Gaussian-Filter
   - Do it by hardware (faster)
3. Convert Image from RGB to gray
4. Approximate gradient
   - Only horizontal change
   - Apply Mask $M = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
5. Find maximum value each row
   - Store points
   - Threshold for max x,y-distance
   - Threshold for min gradient value



Steering Control

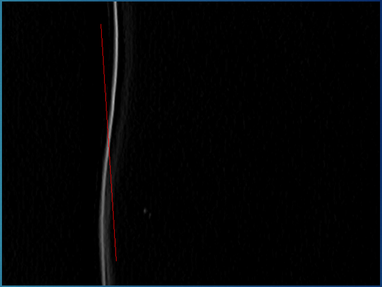
Algorithm



Steering Control

Algorithm

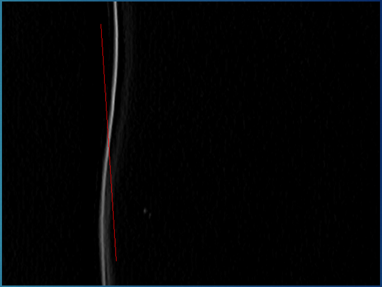
6. Fit line on points (squared distance weighting)



Steering Control

Algorithm

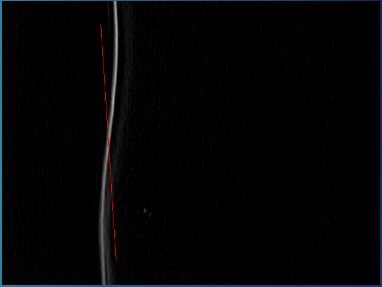
6. Fit line on points (squared distance weighting)
7. Calculate steering angle



Steering Control

Algorithm

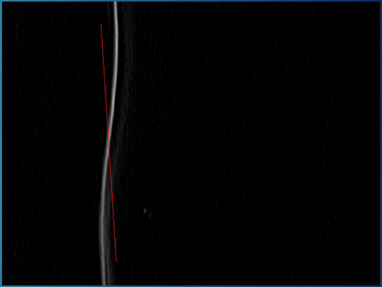
6. Fit line on points (squared distance weighting)
7. Calculate steering angle
8. Add angle:
   $SteeringAngle =$
   $\alpha * NewAngle + (1 - \alpha) * LastAngle$



Steering Control

Algorithm

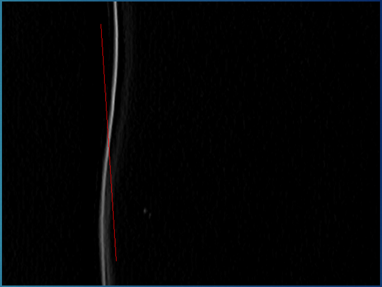
6. Fit line on points (squared distance weighting)
7. Calculate steering angle
8. Add angle: $\mathit{SteeringAngle} = \alpha * \mathit{NewAngle} + (1 - \alpha) * \mathit{LastAngle}$
9. Map angle to servo levels (15 settings)



Steering Control

Algorithm

6. Fit line on points (squared distance weighting)
7. Calculate steering angle
8. Add angle: $\mathit{SteeringAngle} = \alpha * \mathit{NewAngle} + (1 - \alpha) * \mathit{LastAngle}$
9. Map angle to servo levels (15 settings)
10. Send via Uart



Steering Control

Parameters

- Weight of new angle (α)
- Threshold for distance and gradients
- Mapping (linear, progressing)
- Number of threads

Optimization Strategies
Troubleshooting

Optimization Strategies
Troubleshooting

First tests: Not so good...

Optimization Strategies
Troubleshooting

First tests: Not so good...
Why does the car lose the line?

Optimization Strategies
Troubleshooting

First tests: Not so good...
Why does the car lose the line?
Use laptop instead of Raspberry Pi:

Optimization Strategies

Troubleshooting

First tests: Not so good...
Why does the car lose the line?
Use laptop instead of Raspberry Pi:

- Car worked perfectly fine

Optimization Strategies
Troubleshooting

First tests: Not so good...
Why does the car lose the line?
Use laptop instead of Raspberry Pi:

- Car worked perfectly fine
- Reason:

Optimization Strategies
Troubleshooting

First tests: Not so good...
Why does the car lose the line?
Use laptop instead of Raspberry Pi:

- Car worked perfectly fine
- Reason:
  - Computation power

Optimization Strategies
Troubleshooting

First tests: Not so good...
Why does the car lose the line?
Use laptop instead of Raspberry Pi:

- Car worked perfectly fine
- Reason:
  - Computation power
  - $\approx$ 4ms per frame

Optimization Strategies
Troubleshooting

First tests: Not so good...
Why does the car lose the line?
Use laptop instead of Raspberry Pi:

- Car worked perfectly fine
- Reason:
  - Computation power
  - $\approx$ 4ms per frame (Pi: $\approx$170ms)

Optimization Strategies
Troubleshooting

First tests: Not so good...
Why does the car lose the line?
Use laptop instead of Raspberry Pi:

- Car worked perfectly fine
- Reason:
  - Computation power
  - $\approx$ 4ms per frame (Pi: $\approx$170ms)
  - Camera: 15 FPS

Optimization Strategies
Troubleshooting

First tests: Not so good...
Why does the car lose the line?
Use laptop instead of Raspberry Pi:

- Car worked perfectly fine
- Reason:
  - Computation power
  - $\approx$ 4ms per frame (Pi: $\approx$170ms)
  - Camera: 15 FPS $\Rightarrow$ new frame every 66ms

Optimization Strategies
Troubleshooting

First tests: Not so good...
Why does the car lose the line?
Use laptop instead of Raspberry Pi:

- Car worked perfectly fine
- Reason:
  - Computation power
  - $\approx$ 4ms per frame (Pi: $\approx$170ms)
  - Camera: 15 FPS $\Rightarrow$ new frame every 66ms
- BUT: Algorithm concept works!

Optimization Strategies
Guideline

Main Problems:

Optimization Strategies
Guideline

Main Problems:

- Runtime
  - Reaction time ⇔ feasible speed

Optimization Strategies
Guideline

Main Problems:

- Runtime
  - Reaction time $\Leftrightarrow$ feasible speed
- Not all frames of the camera are processed

Optimization Strategies
Guideline

Main Problems:

- Runtime
  - Reaction time $\Leftrightarrow$ feasible speed
- Not all frames of the camera are processed

Main Improvements:

Optimization Strategies
Guideline

Main Problems:

- Runtime
  - Reaction time $\Leftrightarrow$ feasible speed
- Not all frames of the camera are processed

Main Improvements:

- Improve runtime
  - Improve reaction time $\Leftrightarrow$ feasible speed

Optimization Strategies
Guideline

Main Problems:

- Runtime
  - Reaction time ⇔ feasible speed
- Not all frames of the camera are processed

Main Improvements:

- Improve runtime
  - Improve reaction time ⇔ feasible speed
- Process every frame
  - Improve feasible speed

Optimization Strategies
Optimize Runtime

Optimization Strategies
Optimize Runtime

Disabled Gaussian Filter

Optimization Strategies
Optimize Runtime

Disabled Gaussian Filter

- Saves a lot of computation time

Optimization Strategies
Optimize Runtime

Disabled Gaussian Filter

- Saves a lot of computation time
- Replaced it by simple hardware trick

Optimization Strategies

Optimize Runtime

Disabled Gaussian Filter

- Saves a lot of computation time
- Replaced it by simple hardware trick
    - changed focus of camera

Disabled Gaussian Filter

- Saves a lot of computation time
- Replaced it by simple hardware trick
  - changed focus of camera $\Rightarrow$ image gets blurred

Optimization Strategies
Optimize Runtime

Disabled Gaussian Filter

- Saves a lot of computation time
- Replaced it by simple hardware trick
  - changed focus of camera $\Rightarrow$ image gets blurred

Restrict search range

Optimization Strategies
Optimize Runtime

Disabled Gaussian Filter

- Saves a lot of computation time
- Replaced it by simple hardware trick
    - changed focus of camera $\Rightarrow$ image gets blurred

Restrict search range

- Before: search total line for largest Gradient

Optimization Strategies
Optimize Runtime

Disabled Gaussian Filter

- Saves a lot of computation time
- Replaced it by simple hardware trick
  - changed focus of camera $\Rightarrow$ image gets blurred

Restrict search range

- Before: search total line for largest Gradient
- Now: only search at x-Position of last line ± 80 pixels

Optimization Strategies

Optimize Runtime

Disabled Gaussian Filter

- Saves a lot of computation time
- Replaced it by simple hardware trick
  - changed focus of camera $\Rightarrow$ image gets blurred

Restrict search range

- Before: search total line for largest Gradient
- Now: only search at x-Position of last line ± 80 pixels

Only consider Gradients >25

Optimization Strategies
Optimize Runtime

Disabled Gaussian Filter

- Saves a lot of computation time
- Replaced it by simple hardware trick
  - changed focus of camera $\Rightarrow$ image gets blurred

Restrict search range

- Before: search total line for largest Gradient
- Now: only search at x-Position of last line $\pm$80 pixels

Only consider Gradients >25

- If not found in search range: ignore line

Optimization Strategies
exploit concurrency

Optimization Strategies
exploit concurrency

Optimized runtime: $\approx$100-110ms per frame

Optimization Strategies
exploit concurrency

Optimized runtime: ≈100-110ms per frame

- Pretty good, but still every second frame lost

Optimization Strategies
exploit concurrency

Optimized runtime: $\approx$100-110ms per frame

- Pretty good, but still every second frame lost
- We are only using one core

Optimization Strategies
exploit concurrency

Optimized runtime: $\approx$100-110ms per frame

- Pretty good, but still every second frame lost
- We are only using one core, but Raspberry Pi has 4!

Optimization Strategies

exploit concurrency

Optimized runtime: ≈100-110ms per frame

- Pretty good, but still every second frame lost
- We are only using one core, but Raspberry Pi has 4!

Recompile OpenCV WITH_OPENMP

Optimization Strategies
exploit concurrency

Optimized runtime: ≈100-110ms per frame

- Pretty good, but still every second frame lost
- We are only using one core, but Raspberry Pi has 4!

Recompile OpenCV WITH_OPENMP

- Uses OpenMP to parallelize OpenCV calls

Optimization Strategies
exploit concurrency

Optimized runtime: ≈100-110ms per frame

- Pretty good, but still every second frame lost
- We are only using one core, but Raspberry Pi has 4!

Recompile OpenCV WITH_OPENMP

- Uses OpenMP to parallelize OpenCV calls
- Runtime: ≈80ms

Optimization Strategies
exploit concurrency

Optimized runtime: ≈100-110ms per frame

- Pretty good, but still every second frame lost
- We are only using one core, but Raspberry Pi has 4!

Recompile OpenCV WITH_OPENMP

- Uses OpenMP to parallelize OpenCV calls
- Runtime: ≈80ms
- still >66ms

Optimization Strategies
exploit concurrency

Optimized runtime: $\approx$100-110ms per frame

- Pretty good, but still every second frame lost
- We are only using one core, but Raspberry Pi has 4!

Recompile OpenCV WITH_OPENMP

- Uses OpenMP to parallelize OpenCV calls
- Runtime: $\approx$80ms
- still >66ms $\Rightarrow$ Not good enough

Optimization Strategies

exploit concurrency

Optimization Strategies

exploit concurrency

If we can't catch all frames with one thread why not use two?

Optimization Strategies
exploit concurrency

If we can't catch all frames with one thread why not use two?

1. Main thread starts two worker threads and sleeps

Optimization Strategies
exploit concurrency

If we can’t catch all frames with one thread why not use two?

1. Main thread starts two worker threads and sleeps
2. Worker thread grabs a image from the camera (synchronized)

Optimization Strategies
exploit concurrency

If we can’t catch all frames with one thread why not use two?

1. Main thread starts two worker threads and sleeps
2. Worker thread grabs a image from the camera (synchronized)
3. Worker processes the image

Optimization Strategies
exploit concurrency

If we can’t catch all frames with one thread why not use two?

1. Main thread starts two worker threads and sleeps
2. Worker thread grabs a image from the camera (synchronized)
3. Worker processes the image
4. Worker checks whether a later frame had already finished

Optimization Strategies
exploit concurrency

If we can’t catch all frames with one thread why not use two?

1. Main thread starts two worker threads and sleeps
2. Worker thread grabs a image from the camera (synchronized)
3. Worker processes the image
4. Worker checks whether a later frame had already finished
5. Worker wakes Main thread and communicates the calculated direction

Optimization Strategies
exploit concurrency

If we can't catch all frames with one thread why not use two?

1. Main thread starts two worker threads and sleeps
2. Worker thread grabs a image from the camera (synchronized)
3. Worker processes the image
4. Worker checks whether a later frame had already finished
5. Worker wakes Main thread and communicates the calculated direction
6. Main thread generates UART signal and sleeps again

Optimization Strategies
exploit concurrency

If we can’t catch all frames with one thread why not use two?

1. Main thread starts two worker threads and sleeps
2. Worker thread grabs a image from the camera (synchronized)
3. Worker processes the image
4. Worker checks whether a later frame had already finished
5. Worker wakes Main thread and communicates the calculated direction
6. Main thread generates UART signal and sleeps again

This does not affect Reaction time, but increases performance

Optimization Strategies
exploit concurrency

Optimization Strategies
exploit concurrency

Our approach and the WITH_OPENMP option don't work well together

Optimization Strategies
exploit concurrency

Our approach and the WITH_OPENMP option don't work well together

- Runtime ≈120-130ms per frame (using 2 threads)

Optimization Strategies
exploit concurrency

Our approach and the WITH_OPENMP option don’t work well together

- Runtime $\approx$120-130ms per frame (using 2 threads)
- Probably to many threads started for the Pi

Optimization Strategies
exploit concurrency

Our approach and the WITH_OPENMP option don't work well together

- Runtime $\approx$120-130ms per frame (using 2 threads)
- Probably to many threads started for the Pi

Without using multithreading in OpenCV

Optimization Strategies
exploit concurrency

Our approach and the WITH_OPENMP option don’t work well together

- Runtime ≈120-130ms per frame (using 2 threads)
- Probably to many threads started for the Pi

Without using multithreading in OpenCV

- Runtime ≈100-110ms per frame (using 2 threads)

Optimization Strategies

exploit concurrency

Our approach and the WITH_OPENMP option don't work well together

- Runtime $\approx$120-130ms per frame (using 2 threads)
- Probably to many threads started for the Pi

Without using multithreading in OpenCV

- Runtime $\approx$100-110ms per frame (using 2 threads)
- Threads grab frames alternating

Optimization Strategies

exploit concurrency

Our approach and the WITH_OPENMP option don't work well together

- Runtime $\approx$120-130ms per frame (using 2 threads)
- Probably to many threads started for the Pi

Without using multithreading in OpenCV

- Runtime $\approx$100-110ms per frame (using 2 threads)
- Threads grab frames alternating
- Every frame gets processed

Optimization Strategies
summary

Optimization Strategies
summary

Optimizations:

Optimization Strategies
summary

Optimizations:

- Reduce runtime per frame

Optimization Strategies
summary

Optimizations:

- Reduce runtime per frame
- Use multiple threads

Optimization Strategies
summary

Optimizations:

- Reduce runtime per frame
- Use multiple threads
- Ensure every frame of the camera is used

Optimization Strategies
summary

Optimizations:

- Reduce runtime per frame
- Use multiple threads
- Ensure every frame of the camera is used

Result:

Optimization Strategies
summary

Optimizations:

- Reduce runtime per frame
- Use multiple threads
- Ensure every frame of the camera is used

Result:

- Grate performance is achieved

Optimization Strategies
summary

Optimizations:

- Reduce runtime per frame
- Use multiple threads
- Ensure every frame of the camera is used

Result:

- Grate performance is achieved
- Car follows the line

Optimization Strategies
summary

Optimizations:

- Reduce runtime per frame
- Use multiple threads
- Ensure every frame of the camera is used

Result:

- Grate performance is achieved
- Car follows the line
- Increase Speed of the car

Thank you for your attention

Next up: **Live Demo**