

Project Quadcopter

Construction, calibration and modification

Felix Aller <felix.aller@iwr.uni-heidelberg.de>

Roman Hable <romanhable@web.de>

Robert Küchler <kuechler@stud.uni-heidelberg.de>

Edward-Robert Tyercha <e.tyercha@gmx.net>

February 21th 2014

Table of Contents

- ① Hardware and construction — *Robert*
- ② Software, calibration & flying — *Felix*
- ③ Kinect data and object detection as further goal — *Roman*

Goals

Design and construction of a quadcopter



- Further goal: autonomous obstacle detection with Microsoft Kinect

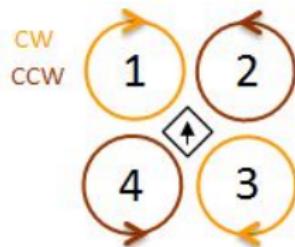
Hardware and construction

Frame

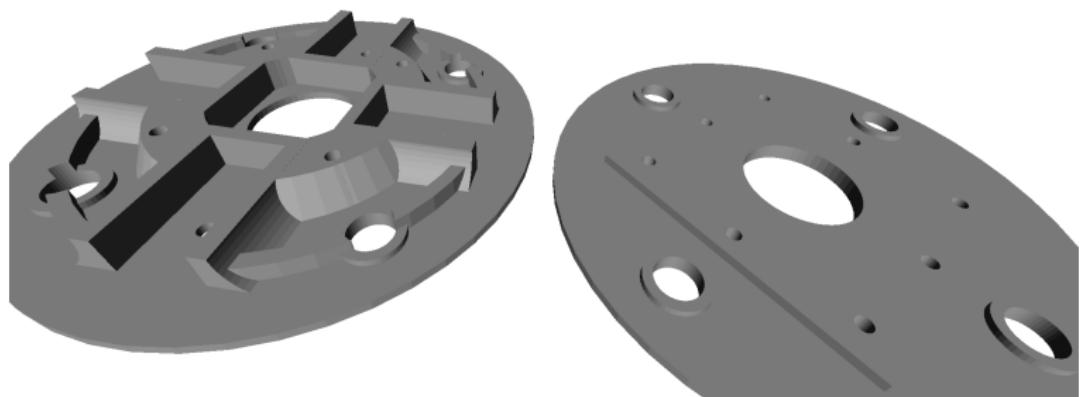
- AeroQuad Cyclone frame



- Quad X configuration



New frameplates 3D-plotted



- Designed new frameplates
- More stability, replaceability

Landing gear

- Temporarily upgraded footings



- New stable gear
- Easy replacable



Motor attachment

- Most replaced part
- Motors need to be parallel to z - axis

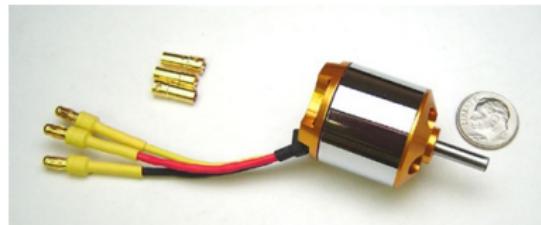


- High heat development
- Washers used to transport heat
- LEDs added for orientation



Motors and Electronic Speed Controllers (ESC)

- 4 x A2217-9 brushless motor
- 950 RPM/V
- 10x4.7 propellers
- 12x4.5 propellers

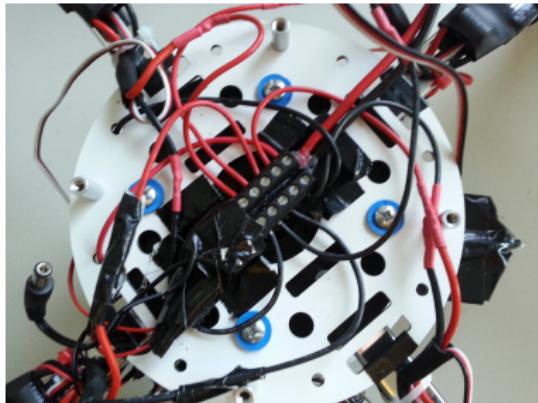


- 4 x brushless flyfun ESC 30A



Battery and power distribution

- 3 cell LiPo battery(11.1V 25C/50C) providing 10min flighttime
- Power distributing to ESCs, LEDs, Arduino board and under voltage protection



- High resistance at clamping block
- Weak connection with tamiya connector
- Changed to traxxas connectors



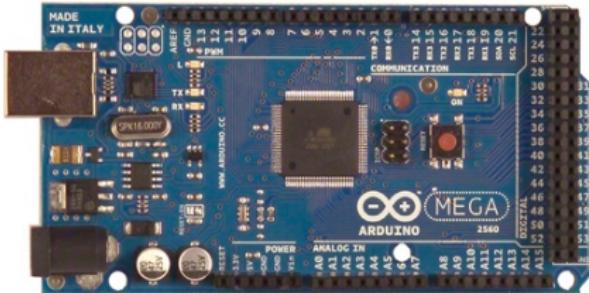
Boards



- Hardware issues with the beagleboard
- Changed to RaspberryPi

Boards(2)

- Arduino Atmega 2560
- Contains the flightsoftware
- With attached gyro, accelerometer, barometer, ultrasound rangesensor
- Wireless communication attached: GPS, Xbee wireless transmitter



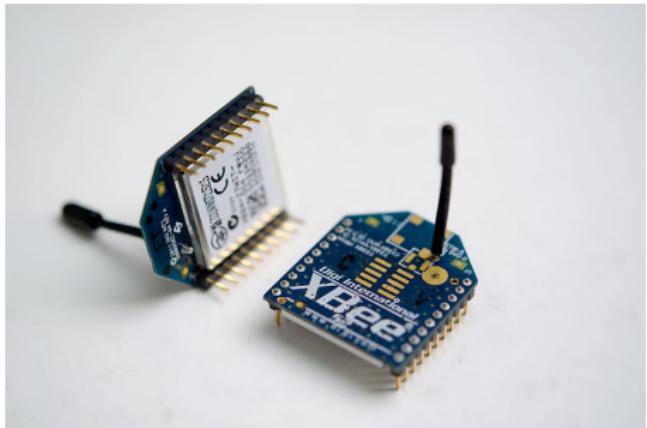
Remote control

- Digital remote control
- Graupner MX-16S
- 8 channel receiver attached to the Atmega board



XBee wireless transmitter

- XBeePro S1
- Second XBee module used with USB dongle for wireless communication



Software, Calibration & Flying

AeroQuad Configurator

AeroQuad Configurator v3.1 - Beta 6

Datei Bearbeiten Connect View Hilfe

Information Display Initial Setup Configuration Altitude Hold

Upload AeroQuad Flight Software

Welcome to the AeroQuad Configurator!

This tool is used to configure, calibrate and test your AeroQuad in preparation for flight. Before using the Configurator, please install the Arduino IDE 1.0 or greater (found at www.arduino.cc) and install the proper drivers for your Arduino board to connect to your computer.

To setup your AeroQuad click on each of the buttons in the Initial Setup screen:

- Upload Flight Software (advanced users can also upload directly from the Arduino IDE)
- Initialize EEPROM (returns user configurable values to their default values)
- Gyroscope Calibration
- Accelerometer Calibration
- Magnetometer Calibration (only valid with hardware configurations supporting this)
- Transmitter Calibration
- Electronic Speed Controller Calibration (perform this calibration with no propellers attached)

After your AeroQuad has been properly setup, you can revisit the Initial Setup screen periodically to update your AeroQuad configuration and calibrations as necessary.

For additional help please visit www.AeroQuad.com to access our wiki documentation, help forums, and IRC chat. Please practice safe flying, as multicopters can be dangerous if not flown with common sense. Do as much testing as possible with the propellers not attached and only fly in approved R/C locations.

Upload to AeroQuad

Software Version	3.1
Board Type	STM32
Flight Config	Quad X
Receiver Channels	8
Motors	4
Gyroscope	Detected
Accelerometer	Detected
Barometer	Detected
Magnetometer	Detected

Connected to the AeroQuad! Com Port: COM4 Baud Rate: 115200 Boot Delay (s): 0 Connect Disconnect

Flight Modes

1. Rate Mode

Uses only gyro data to stabilize

2. Attitude Mode

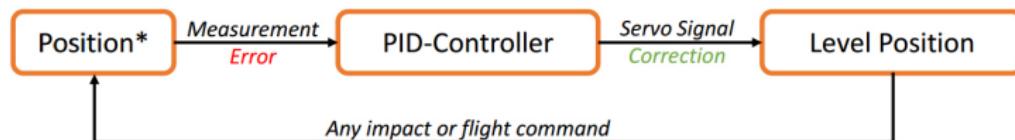
Uses gyro and accelerometer data to maintain level position
→ "auto level" ability

3. Altitude Hold

Maintain the current distance to the ground using ultrasound and barometer

PID controller

- Proportional-Integral-Derivative controller is a generic control loop feedback mechanism
- Calculates error value as difference between measurement and desired setpoint
- Attempts to minimize that error



The Proportional

$$y(t) = K_p \cdot e(t)$$

$e(t)$:= error

K_p := P-gain

$y(t)$:= correction

- Most obvious control parameter
- Calculates error at time t
- Gain value controls sensitivity of correction

- Too big values: copter starts oscillating
- Too low values: copter reacts too slow

The Integral

$$y(t) = K_I \cdot \int_0^t e(r) dr$$

$e(t)$:= error

K_I := I-gain

$y(t)$:= correction

- 'Slowest' control parameter
- Utilizes errors of the past to prevent errors in the present
- Can eliminate the Steady-State-Error
- Is somehow buggy in the AeroQuad Software

The Differential

- Guesses the error in the future

$$y(t) = K_D \cdot \frac{d}{dt} e(t)$$

$e(t)$:= error

K_D := D-gain

$y(t)$:= correction

$e'(t)$ assumes:

If the error in $[t - 1, t]$ is decreasing

It will most likely decrease in
 $[t, t + 1]$

- We did not really modify or experiment with the standard D-gain

PID controller

PID-Controller = P + I + D, or

$$y(t) = K_p \cdot e(t) + K_I \cdot \int_0^t e(r) dr + K_D \cdot \frac{d}{dt} e(t)$$

But how did we manage to find out the correct gain parameters?

PID controller

PID-Controller = P + I + D, or

$$y(t) = K_p \cdot e(t) + K_I \cdot \int_0^t e(r) dr + K_D \cdot \frac{d}{dt} e(t)$$

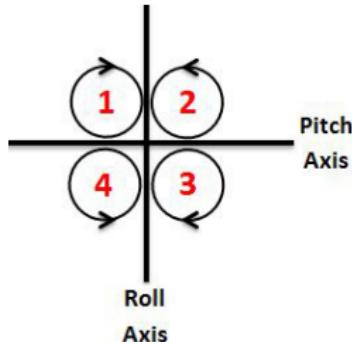
But how did we manage to find out the correct gain parameters?

We guessed calibrated them by in/decreasing over and over again

Problems

- AeroQuad configurator is really really bad
- PID-Calibration is not self-explanatory and not well documented
- Configurator produces random values
- Can fill random values in random forms (looks intended)
- Save and Recall PID-calibration file function broken
- Only working thing is the upload button
 - New python software to up/download calibration
 - Even more functionality e.g. parameters to tune altitude hold

Flying - Basics



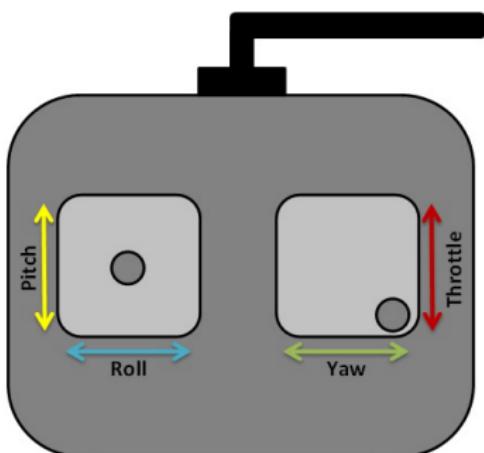
Raise ↑ 1,2,3,4	Sink ↓ 1,2,3,4	Left(Yaw) ↑ 2,4 ↓ 1,3	Right(Yaw) ↑ 1,3 ↓ 2,4
Front(Pitch) ↑ 3,4 ↓ 1,2	Back (Pitch) ↑ 1,2 ↓ 3,4	Left(Roll) ↑ 1,4 ↓ 2,3	Right(Roll) ↑ 2,3 ↓ 1,4

Flying - Controls

- Controlling all this by hand is impossible
- Reason that quadcopters became popular since 2000
 - High performance microcontrollers with gyroscope support make navigating possible

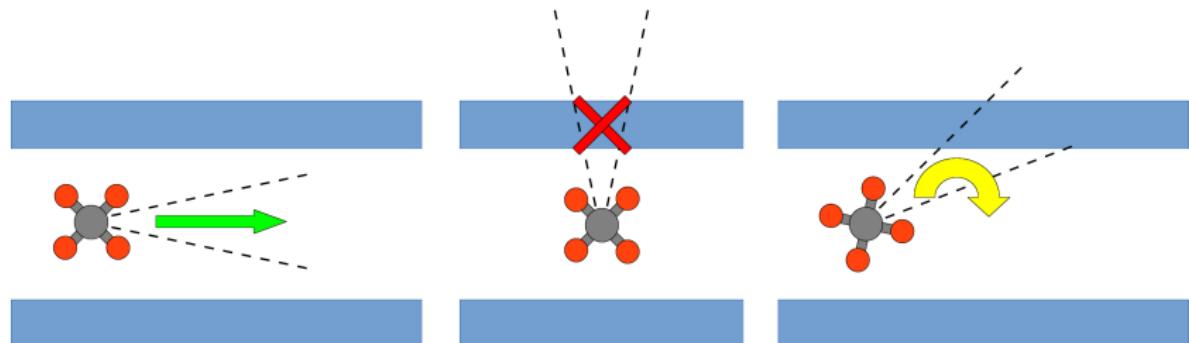
In addition:

- Strong lithium polymer batteries
- Availability of small brushless motors
 - better power-to-weight ratio



Kinect data and object detection as further
goal

Autonomous Obstacle Detection: Introduction



- Use sensor data (Kinect) to detect nearby obstacles (e.g. walls)
- Automatically avoid collision with them: stop or correct flight

Considerations

- Safety: It can be turned off at any time (\Rightarrow manual control)
- Performance: The software has very limited time to decide
- Extensibility: Other programs might want to use the sensor data, too

Autonomous Obstacle Detection: Approach

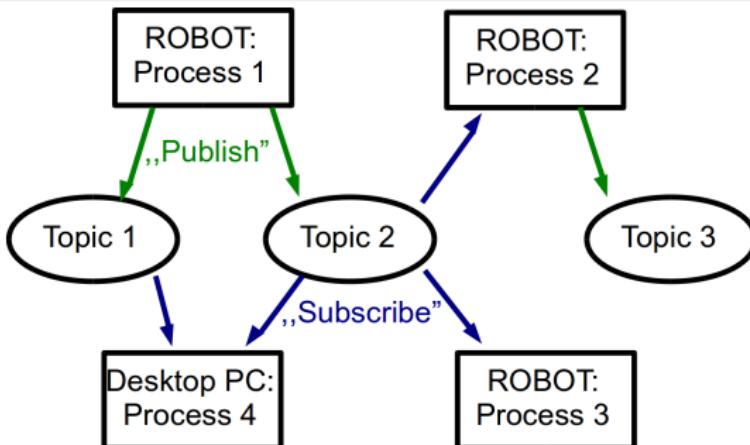
- Start with a simple, efficient prototype 1 (forward or stop)
- When things go well, implement more accurate algorithm 2
⇒ e.g. for corrections, planning ahead, temporal reasoning, ...

Benefits of this approach

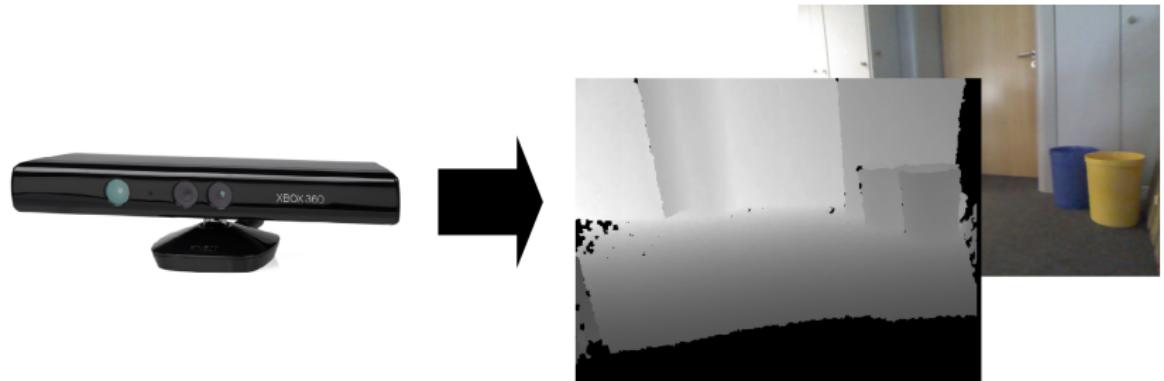
- 1 as emergency stop without unexpected effects/bugs
- Sanity check for reference (to debug 2)
- Gaining knowledge, to find out which algorithm 2 will work out:
 - How far can we go? (Performance evaluation)
 - How does the input look like (size, distribution, extrema)?
 - Which (optimised) functions/algorithms are already supported?

Framework of choice: ROS

- **Robot Operating System**
- Various features, e.g. 3D visualisation of sensor data
- "Topics": message passing between processes (1 in, 0..N out)



Kinect

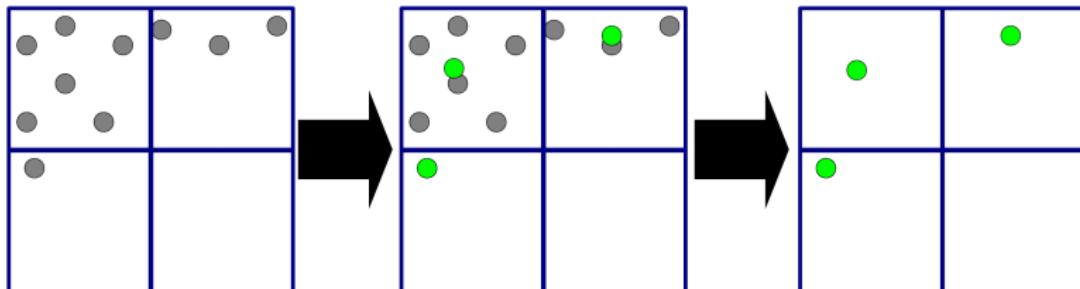


Microsoft Kinect

- RGB camera (640x480, 30 Hz)
- Depth camera (for "Clouds" of 3D-points)
- Accessable by ROS using OpenNI (Open source SDK)
- ROS has a **Point Cloud Library**

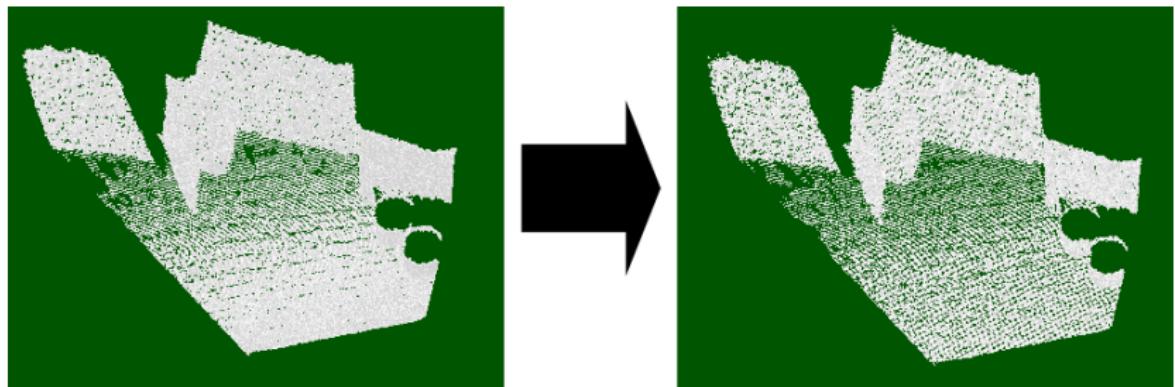
Pointclouds and Downsampling

- Issue: analysing many points = bad performance



- Solution: combine similar points into one
- Downsampling (PCL): use centroids (= means) of Voxel grid cells

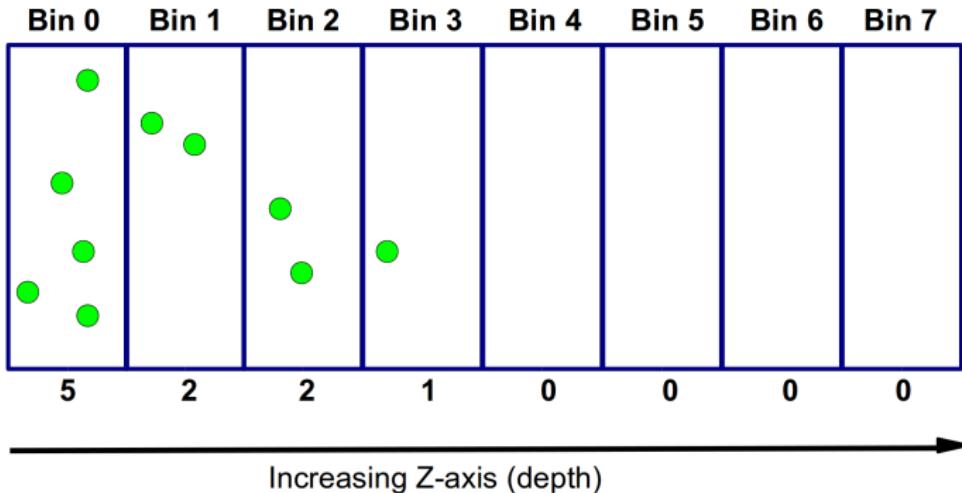
Downsampling: Example



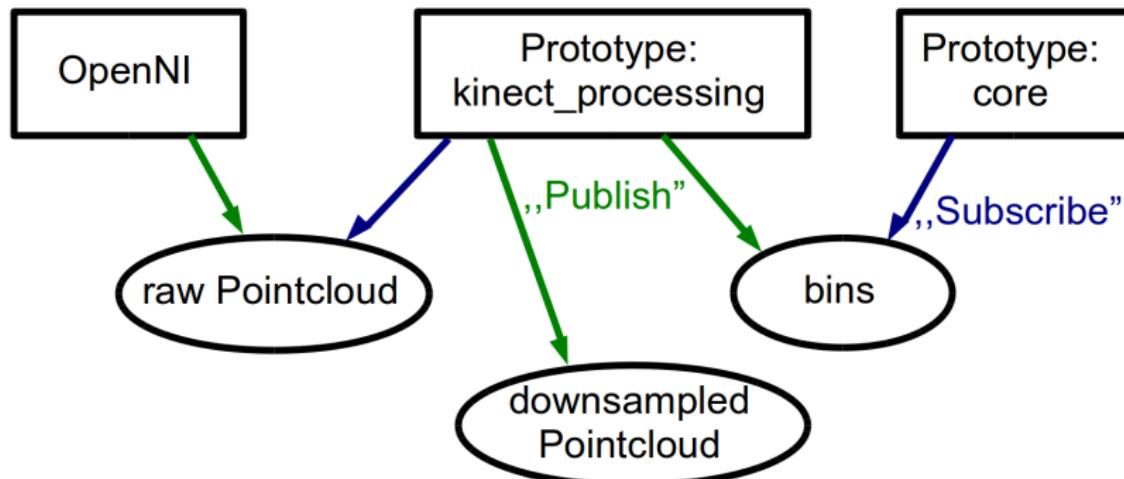
- Shapes remain intact
- Here: ca. 250.000 points reduced to ca. 74.000 points
- About 70% of the original points have been discarded

Our Prototype

- Applies Downsampling
- Accumulates points with similar depth into the same "bins"
- Computes extrema for all dimensions (feel for data)
- Avoids floating-point operations: slow on RaspberryPi



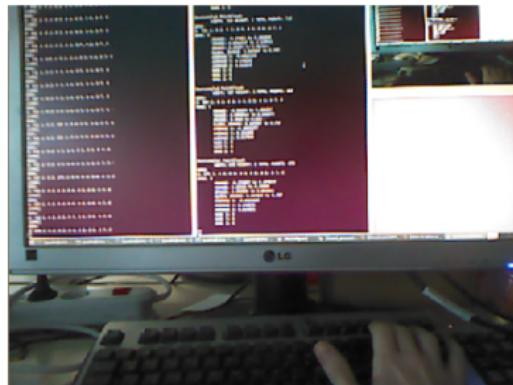
Our Prototype: Architecture



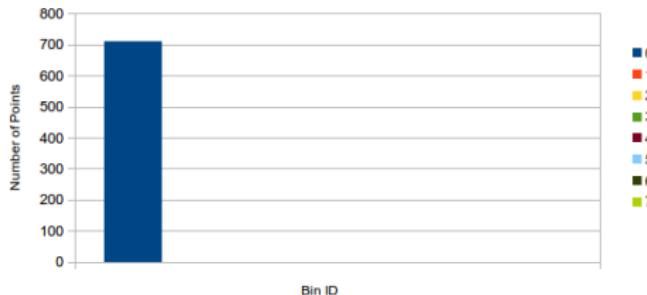
Core:

- responsible for steering decisions
- can iterate much faster than kinect_processing
- could incorporate other sensor data and algorithm results

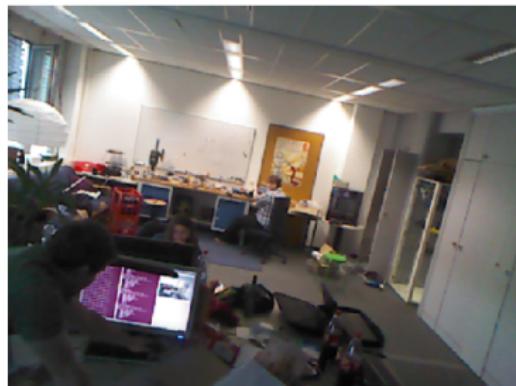
Our Prototype: Practical examples (1/5)



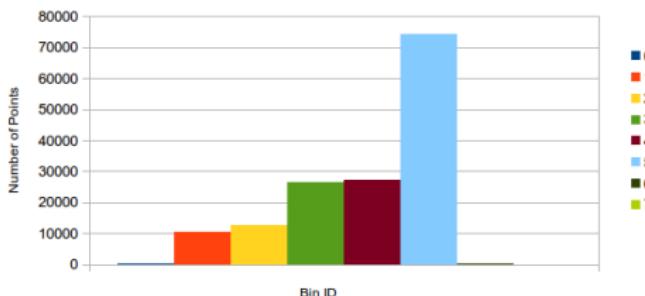
Example 1: ~700 Points



Our Prototype: Practical examples (2/5)



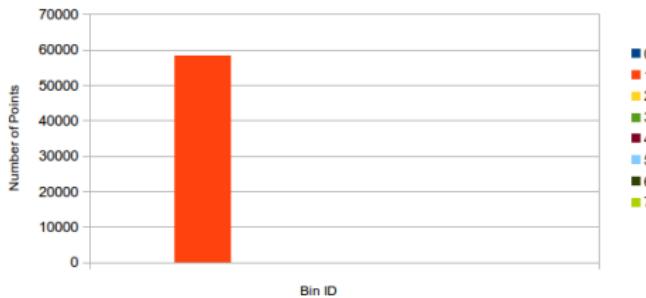
Example 2: ~150.000 Points



Our Prototype: Practical examples (3/5)



Example 3: ~58.000 Points

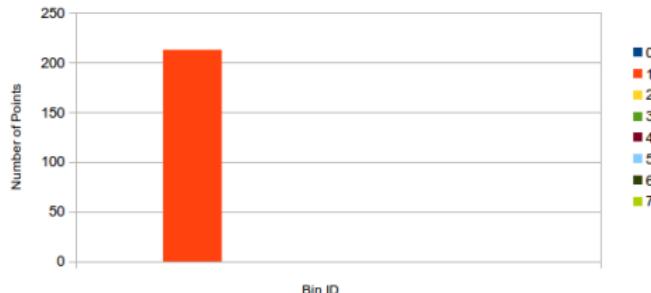


- "Camera-beams" are radial: dense points on objects which are closer
- No matter how dense a grid-cell is: only 1 centroid
- Close objects \Rightarrow fewer points

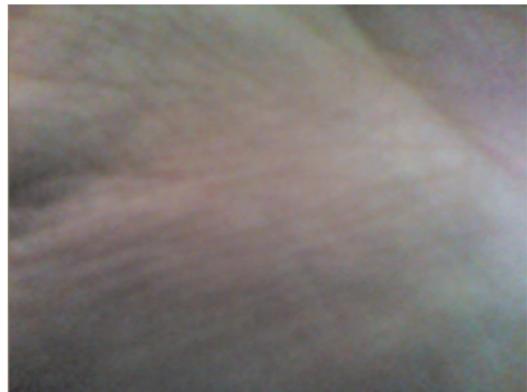
Our Prototype: Practical examples (4/5)



Example 4: ~200 Points



Our Prototype: Practical examples (5/5)



- Zero points in the Pointcloud
- ⇒ there is a minimum distance

Our Prototype: Findings

- Few points indicate close objects
 - "Holes" due to minimal distance
 - Close objects → dense points → few centroids
- Downsampling helps a lot, but takes ca. 50ms (Desktop PC!)
- Bounding Box (global extrema over recording time):
 - x, y : roughly -5 to $+5$ each
 - z : 0.3915 to 9.75 (\Rightarrow used binsize: $\frac{10}{8} = 0.125$)

Our Prototype: Problems

- Many NaN-values for Point-coordinates
 - Filtered out by downsampling
 - Manual filtering: $(v \neq v)$
- Unstable indoor flight (until recently)
- OpenNI not working with ROS on RaspberryPi!

⇒ stopped further development and put more focus on hardware

- Optimize Downsampling for RaspberryPi
 - Avoid floating point operations
 - Think of other methods
- Try out other sensors/software
- If things work out: implement more advanced algorithms
- Improve hardware even further

Thank You!