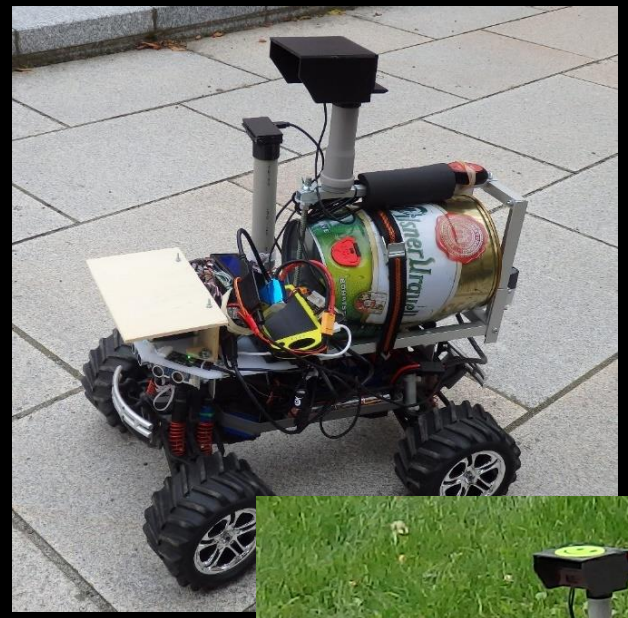
Istrobotics

Robotour 2020, 3.2.2021

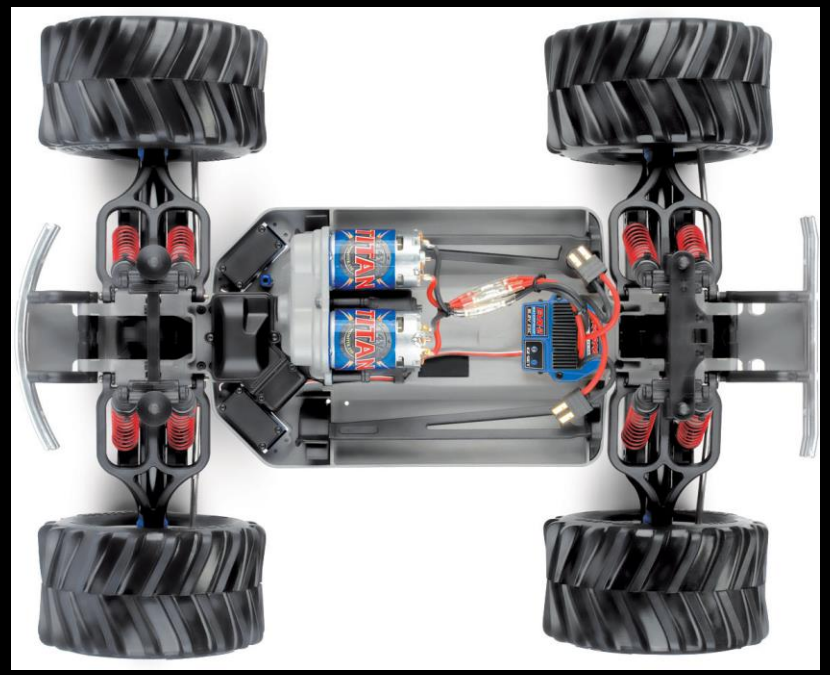
Pavol Boško, Radoslav Kováč

2007 2008 2009 2014 2015

2016 2017 2018 2019 2020

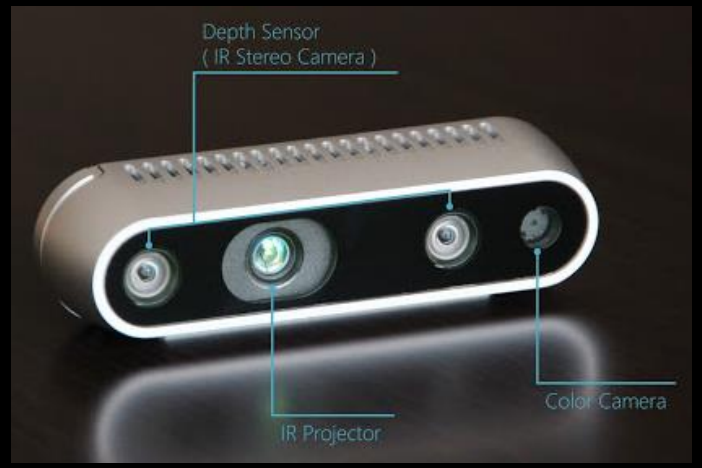


ROBOT CHASSIS



- RC model: Traxxas E-Maxx 4x4 monster truck
- Top Speed: 48 km/h
- Waterproof electronics, servos

HARDWARE

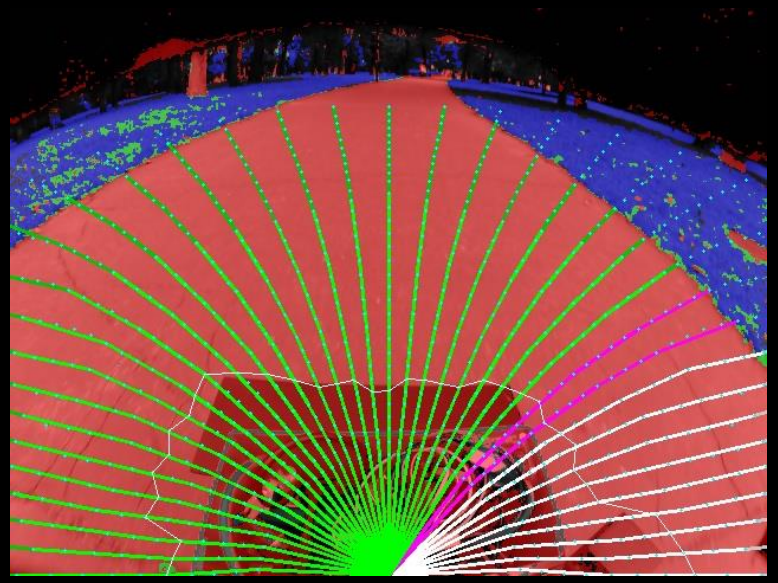


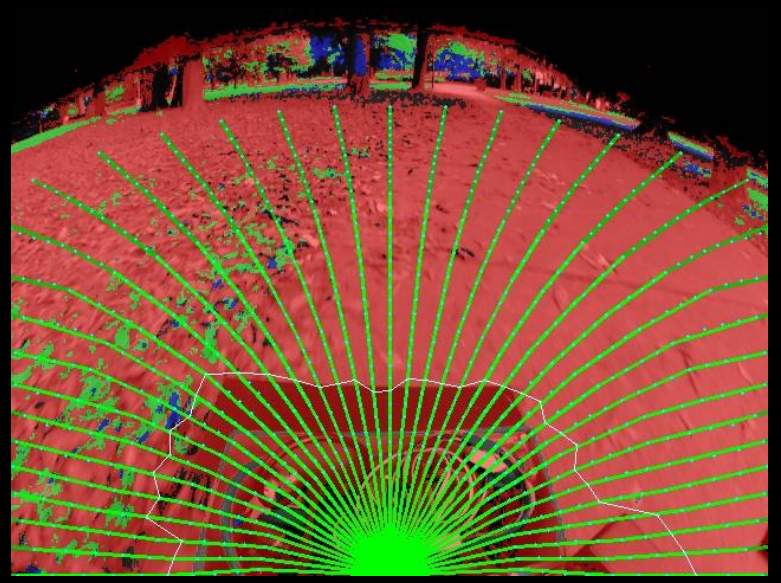
HW Component	Price
Chassis: Traxxas E-Maxx 3903 • Two Speed Conversion Kit, Gear, Shock Set...	440 Eur
Jetson Nano: 1.4GHz 4-core, 4GB RAM, 64GB SD • Arduino Mega: 16MHz, 8KB RAM • 2x Arduino Nano: 16MHz, 2KB RAM • Arduino Pro Mini: 16MHz, 2KB RAM	200 Eur
Depth Camera: Intel RealSense D435 • 2019: Odroid oCam 5MP (640x480, 170 FOV) • 2018: Odroid USB Cam (640x480, 65 FOV)	260 Eur
2D Lidar: RoboPeak RPLIDAR 360, 7Hz, 1° res. • 2019: Sick TiM571, 15Hz, 0.3° res. ($2400)	100 Eur
GPS/Glonass receiver: Holux M-215+	40 Eur
Compass: Bosch BNO055	40 Eur
Baterries: 2x Antix HV-LiPo 9000 mAh, 7.6V	160 Eur
Ultrasonic Distance Sensor: 5x Sonar: HC - SR04	10 Eur
Other: LCD & OLED displays, 8x LED, Wireless NIC...	50 Eur
TOTAL	**1 300 Eur**

VISION (2016 – 2019)
Pixel color classification

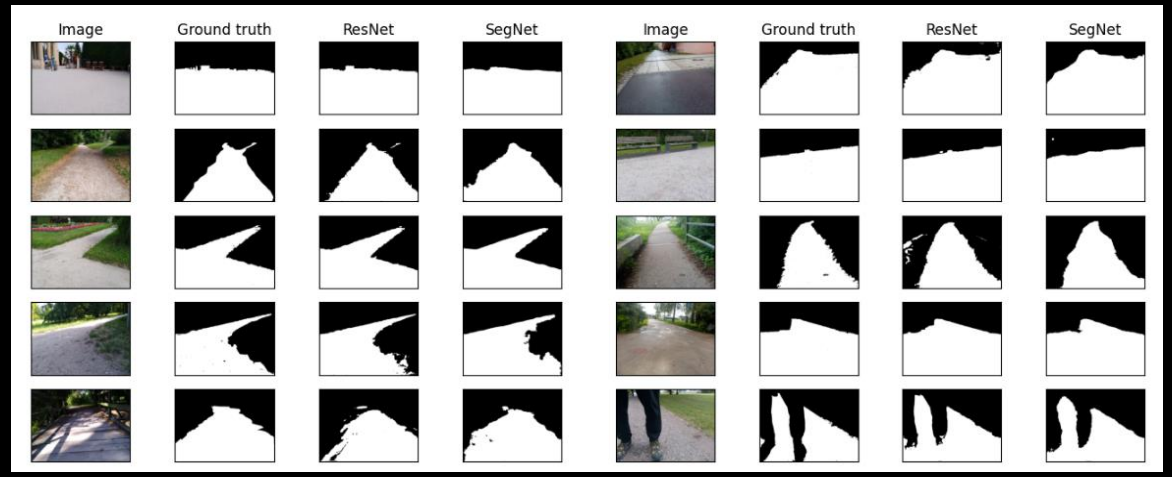
- Our old approach:
  - Classify every pixel as: road / off-road (grass)
  - Manually chosen samples (32x32 pixels)
  - Find nearest cluster in HSV color space
  - Lookup-table could be used: RGB -> class
- Disadvantages:
  - Adding more samples to improve the result was more and more challenging
  - brown leaves (in autumn) could not be distinguished from road

…IMAGE SEQUENCE #1





NEURAL NETWORKS at Robotour

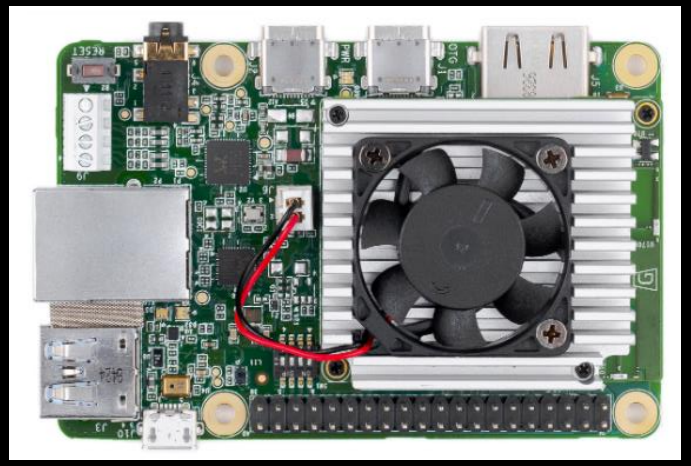


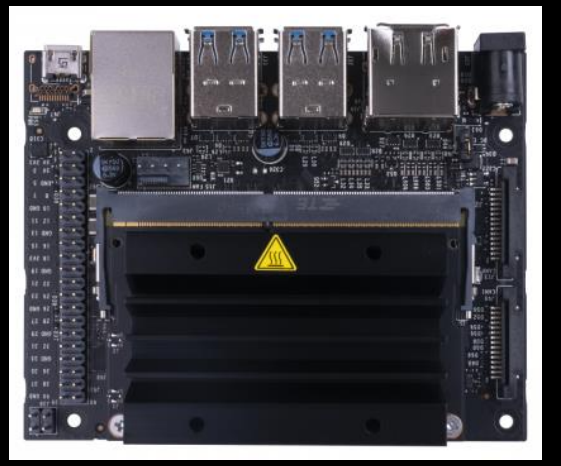
- 2014 – Smelý Zajko (SK), Plecharts (CZ)
  - first robots with neural networks
- 2016 – JECC (DE)
  - first convolutional neural network (CNN)
- 2019 – Smelý Zajko (SK)
  - https://github.com/Adman/road-segmentation
  - Adrián Matejov, Master's Thesis 2020 [1]
  - Jetson TX2: Tensorflow, Keras, Python
  - ResNet: 2,7mil params (33 MB), 240ms prediction
  - 2 datasets: 677 images, Lednice + Deggendorf
- 2020 – we used NN from Smelý Zajko
  - Jetson Nano is 2x slower than Jetson TX2
  - ResNet (f=4): 179.297 params, 200ms prediction
  - TCP between Python server and C++ client

AI ACCELERATOR HARDWARE – Google, Intel and NVIDIA [2]

Board	MobileNet v1 (ms)	MobileNet v2 (ms)	Idle Current (mA)	Peak Current (mA)	Price (US$)
Coral Dev Board	15.7	20.9	600	960	$149.00
Coral USB Accelerator	49.3	58.1	470	880	$74.99+$35.00
NVIDIA Jetson Nano (TF)	276.0	309.3	450	1220	$99.00
NVIDIA Jetson Nano (TF-TRT)	61.6	72.3			
Movidius NCS	115.7	204.5	500	860	$79.00+$35.00
Intel NCS2	87.2	118.6	480	910	$79.00+$35.00
MacBook Pro[1]	33.0	71.0	1570	1950	>$3,000
Raspberry Pi	480.3	654.0	410	1050	$35.00

[1] The MacBook Pro takes a +20V supply, all other platforms take a +5V supply.



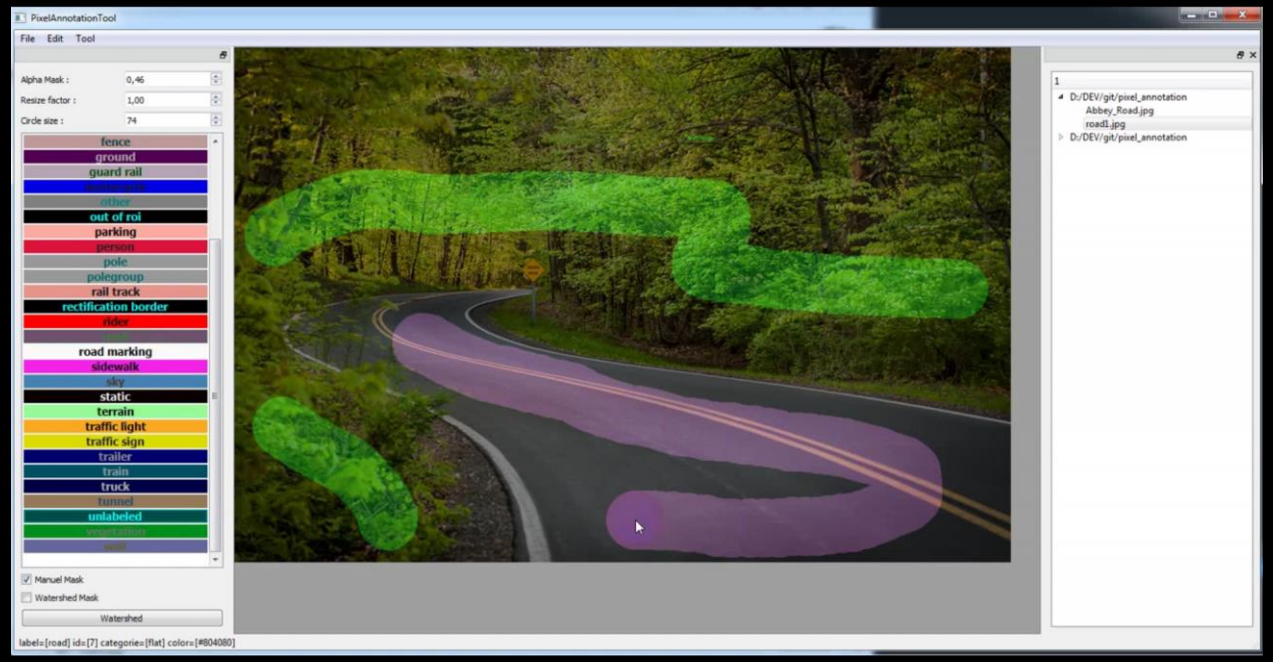


IMPROVING THE DATASET




- What to improve?
  - Imperfect predictions on training images
  - grass-only images are missing in training set
  - images with walls are rare
  - test in different parks / locations / season
- Our database of driving data (logs, images)
  - 14 different parks (7x Bratislava, 3x CZ, 1x DE)
  - 50 driving days, 400 attempts
  - 500.000 images, 220GB data
- Let the NN process images from past competitions
  - then manually select images with bad predictions
  - manually label / annotate new images
  - start NN training again

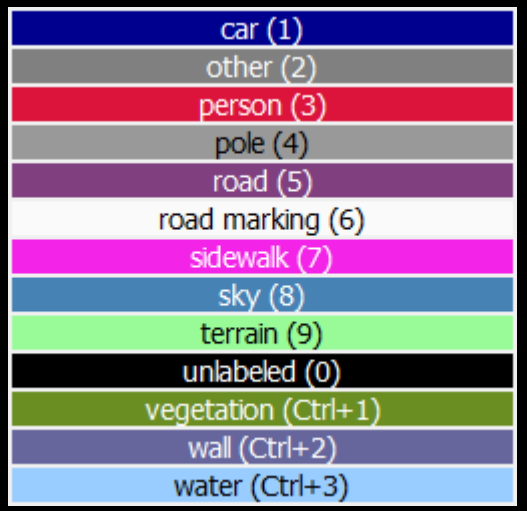
LABELING - PixelAnnotationTool

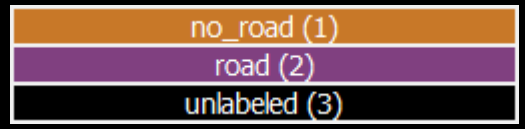


- PixelAnnotationTool
  - manually provide markers with brushes
  - then launch the algorithm
  - refine/correct the markers
  - markers are stored as separate images
- Implements OpenCV watershed algorithm
  - marker-based image segmentation
- https://github.com/abreheret/PixelAnnotationTool
- license: GNU GPLv3

HOW TO ANNOTATE?

- More categories – more time for labeling
  - 2 categories: 90-120 sec / image
  - 10 categories: 180-600 sec / image
- What is the definition of “road”?
  - how about stairs? holes? filled with water?
  - curbs? looking from top to bottom?
  - definition: drivable from the robot position(?)





OUR TRAINING DATASET



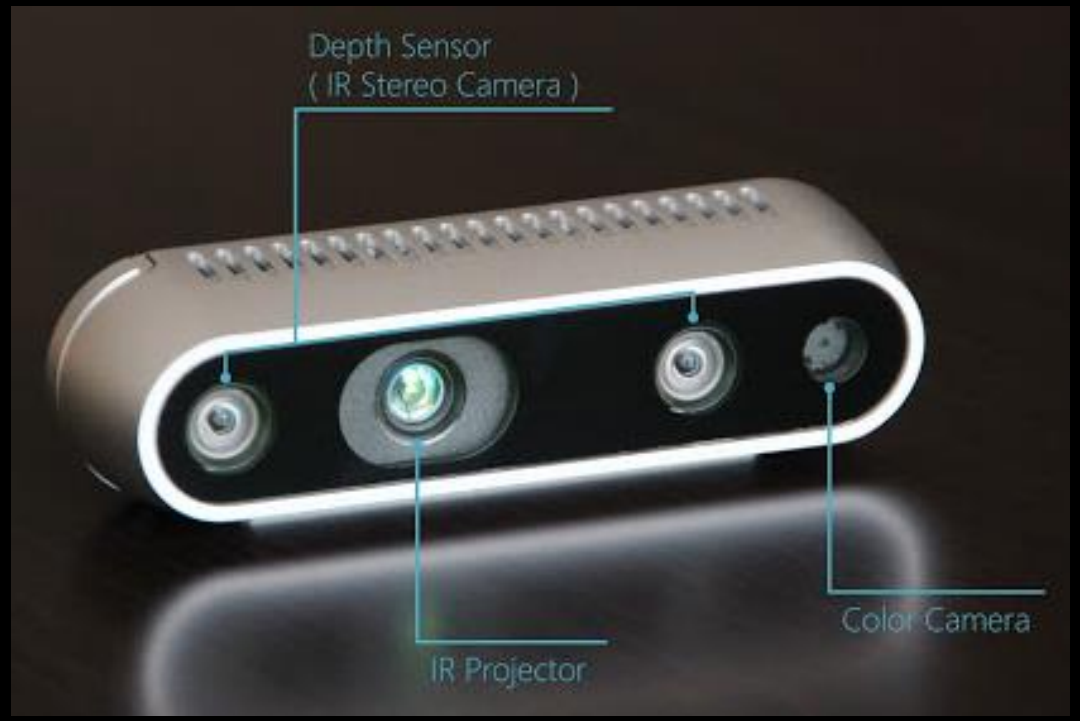



- We annotated 540 new images (road/noroad):
  - CZ, Písek, Palackého Sady: 103 images
  - SK, Bratislava, Sad Janka Kráľa: 206 images
  - SK, Bratislava, Park Andreja Hlinku: 106 images
- https://github.com/lnx-git/istro-rt
  - dataset\image, dataset\masks
- incl. 2 datasets created by Smelý Zajko (677x)
- Data augmentation for NN training
  - blur
  - contrast
  - horizontal flip
  - cropping: scale 3/4 + shift (top/bottom, left/right)
  - grayscale
  - positive effect – needs to be further evaluated [3]

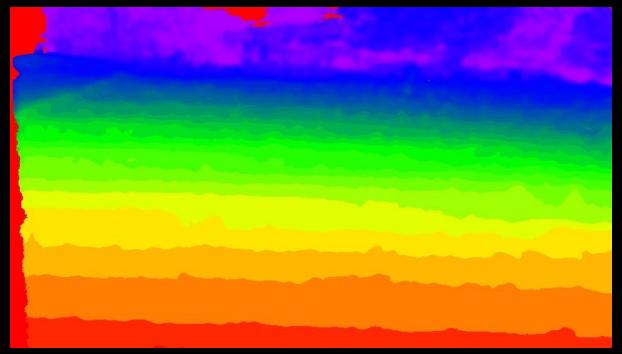
NEURAL NETWORK RESULTS

- Works nice
  - autumn colors are no longer a challenge
  - good(?) performance with grayscale images
- Ready for "Klondajk" (Robotour 2020)?
  - problems with grass on concrete roads
- Grass covered with snow?
  - No way – snow makes it impossible to find road
  - additional training images will be needed

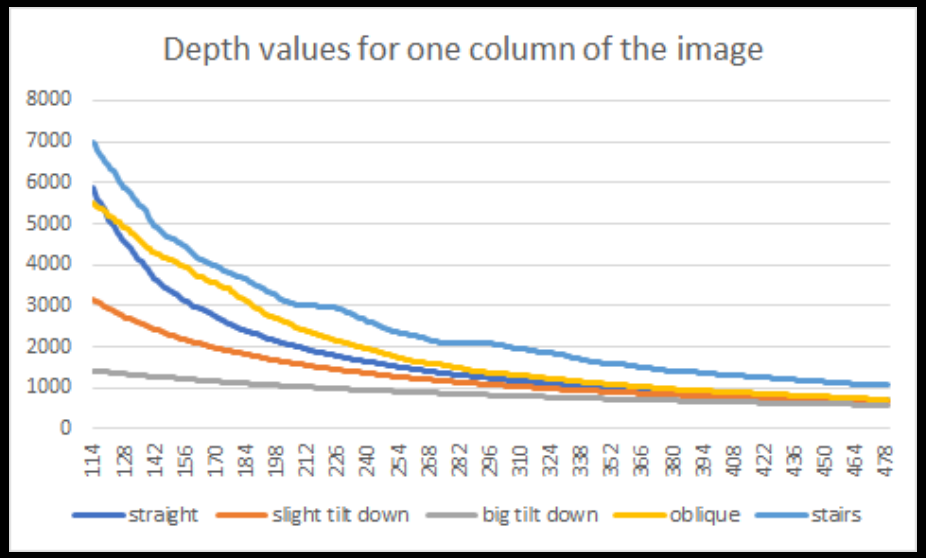
DEPTH CAMERA: Intel RealSense D435

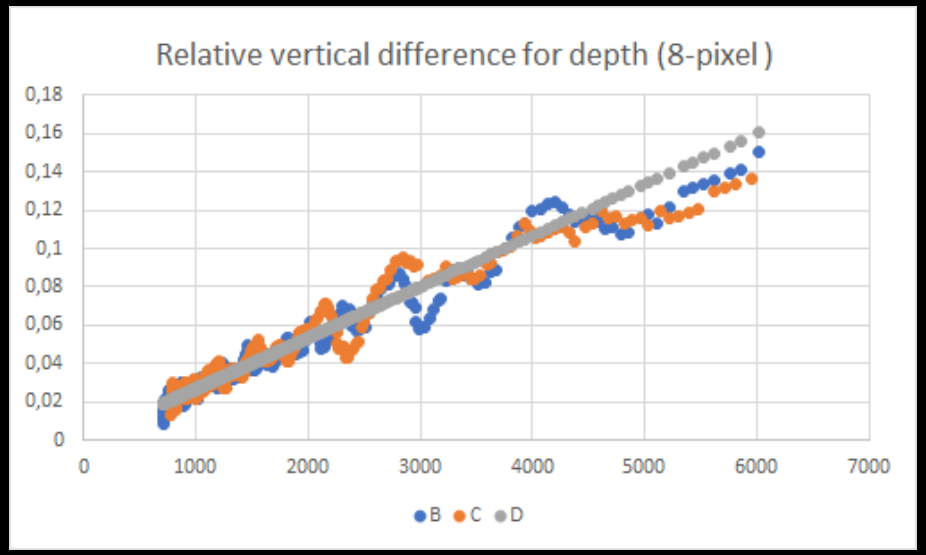


- Intel RealSense D435 features
  - Color Camera Horizontal FOV: 69°, 1920x1080
  - Depth HD Horizontal FOV: 87°, 1280x720
  - IR Projector - to improve depth accuracy in scenes with low texture
- We process images in these resolutions:
  - RGB: 640x480
  - Depth: 848x480
- Other depth camera alternatives:
  - Stereolab ZED Mini, 400+ Eur, includes precise IMU for positional tracking (Smelý Zajko)

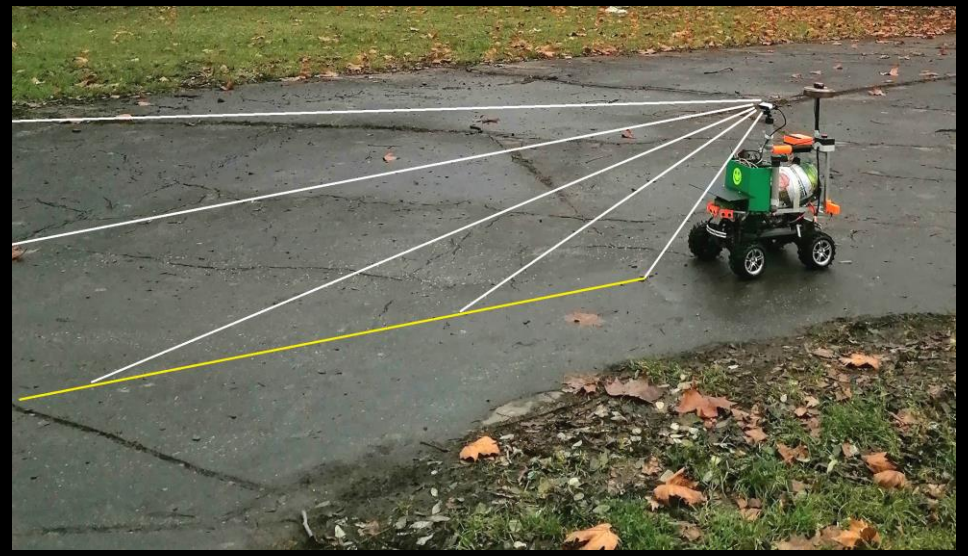


DEPTH IMAGE PROCESSING

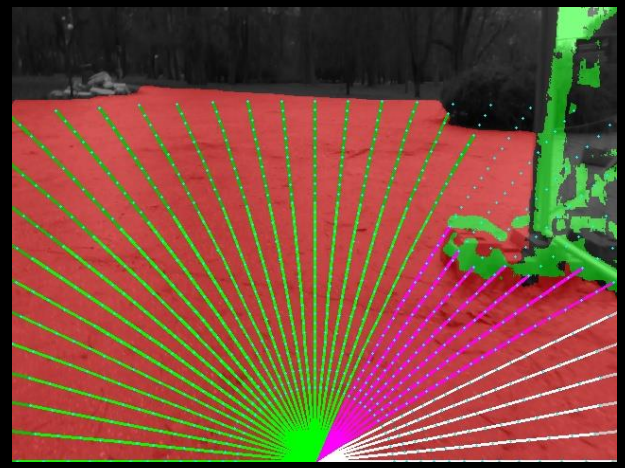
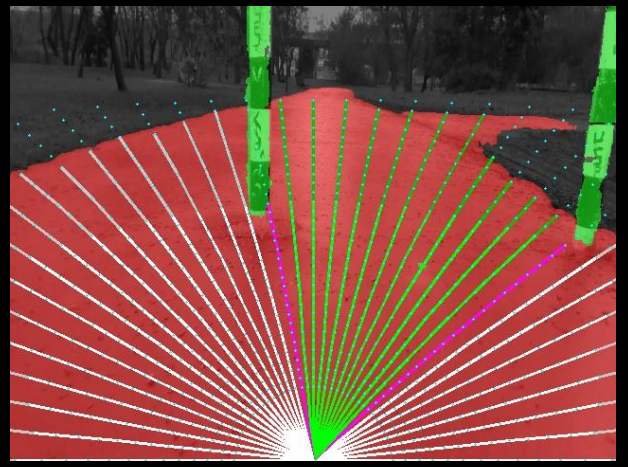
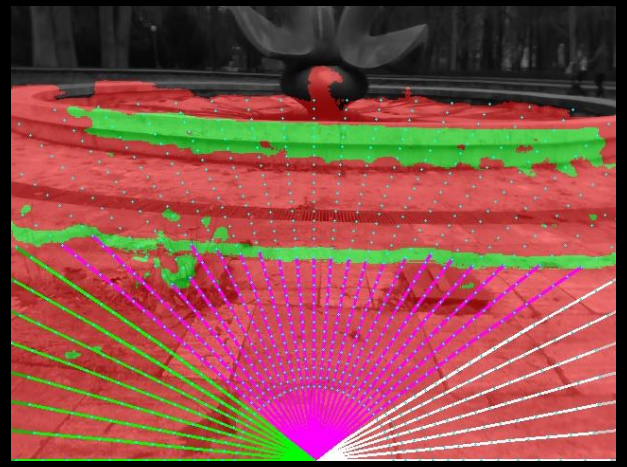


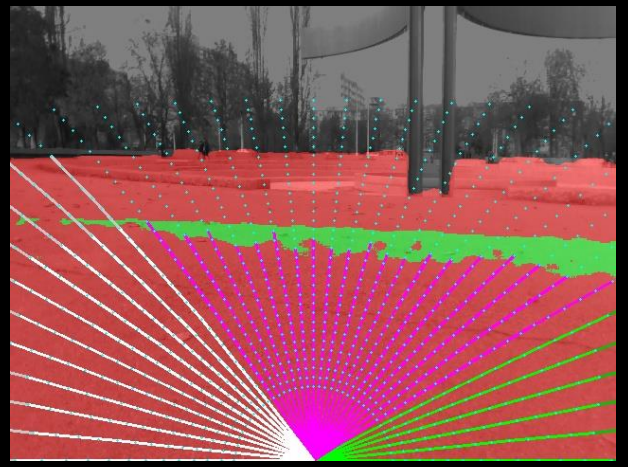


- Detecting obstacles and holes in the road
  - take two pixels: (x,y) and (x, y+8)
  - calculate the difference between the depth values of these two pixels
  - divide by the depth value of the second pixel
  - compare with min and max linear boundary
- Processing time (848x480): 45 ms / image

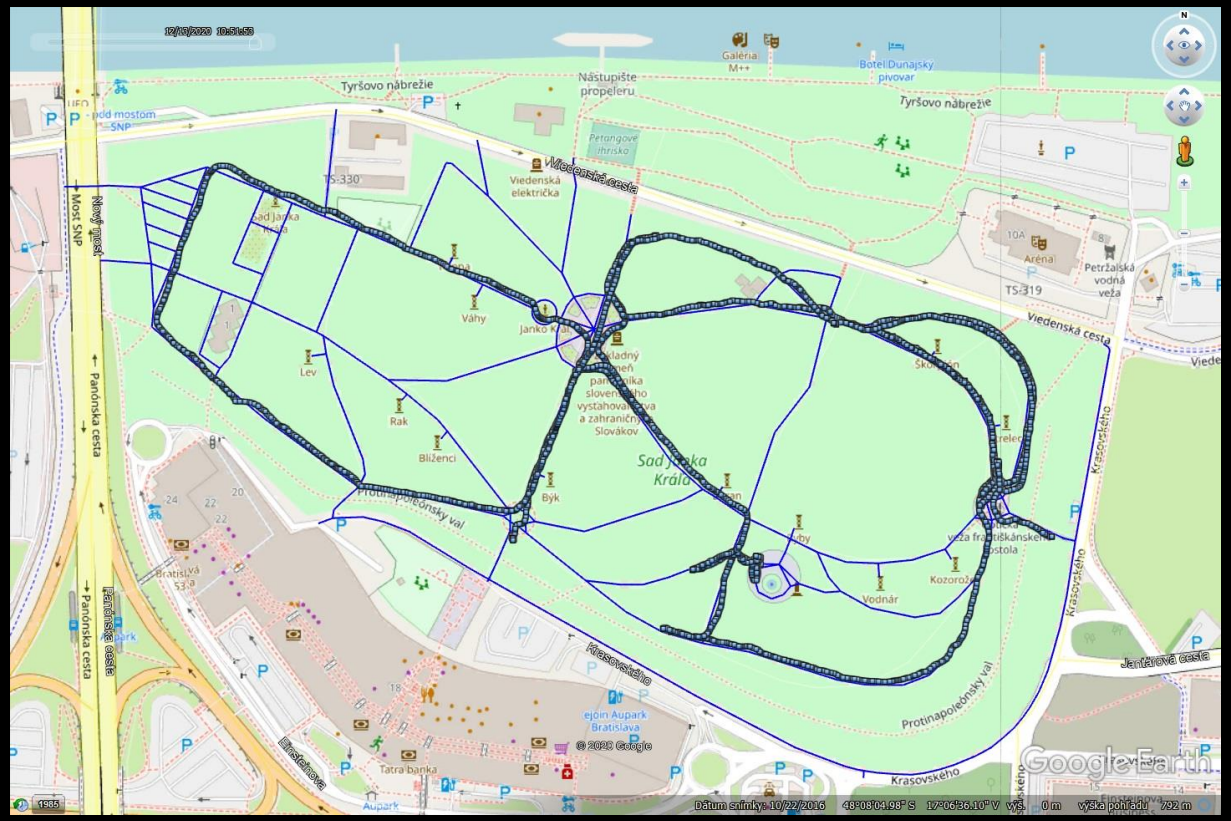


…IMAGE SEQUENCE #2

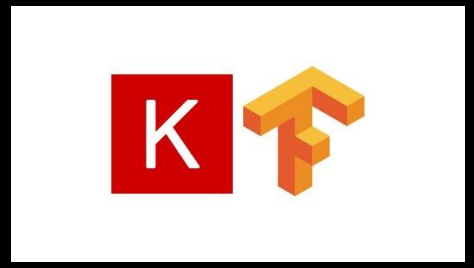


Robotour Marathon – 13.12.2020, 3.95km



- In 2020 we finally implemented
  - wrong-way behavior
  - full-stop (based on vision)
- Problems
  - wrong turns at crossroads
  - sun and water are causing image processing issues
  - roads made of concrete panels
  - grass paver system
- High-capacity batteries are needed to drive in winter
  - we also have a tablet in case the laptop's battery runs out

SOFTWARE



- Operating system: Ubuntu 18.04 (Linux4Tegra)
- Source codes: C++, 707kB
  - 2019: 664kB, 2018: 430kB,
  - 2017: 340kB, 2016: 180kB
- NN: Tensorflow, Keras, Python
- Libraries:
  - OpenCV (vision), GeographicLib (Geo),
  - Zbar (QR-Codes), Libxml2 (.osm),
  - log4cxx (logging)
- Main application + 8x pthreads
  - 4x sensors (Camera, Lidar, GPS , Compass)
  - image capturing + vision processing
  - output: image saving (1GB of data/ round)
  - control board (Compass)

SOURCE CODES

- Sources codes are available at GitHub as public project **Istro RT:**

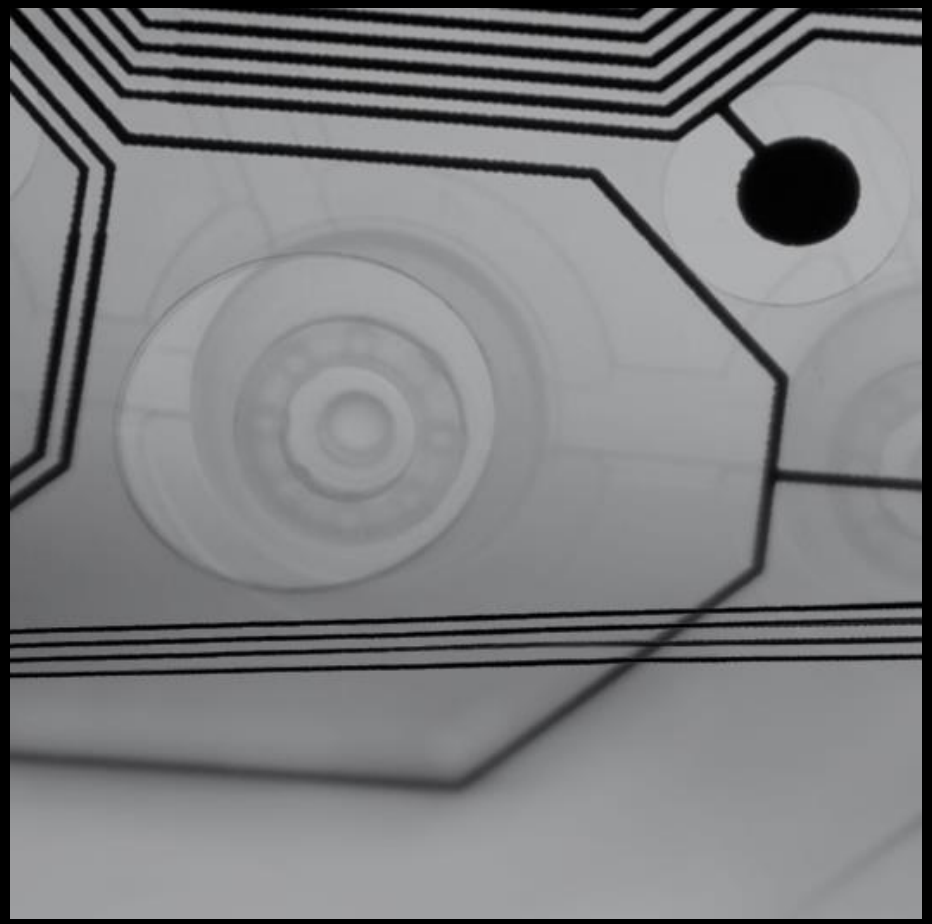
  https://github.com/lnx-git/istro-rt

REFERENCES

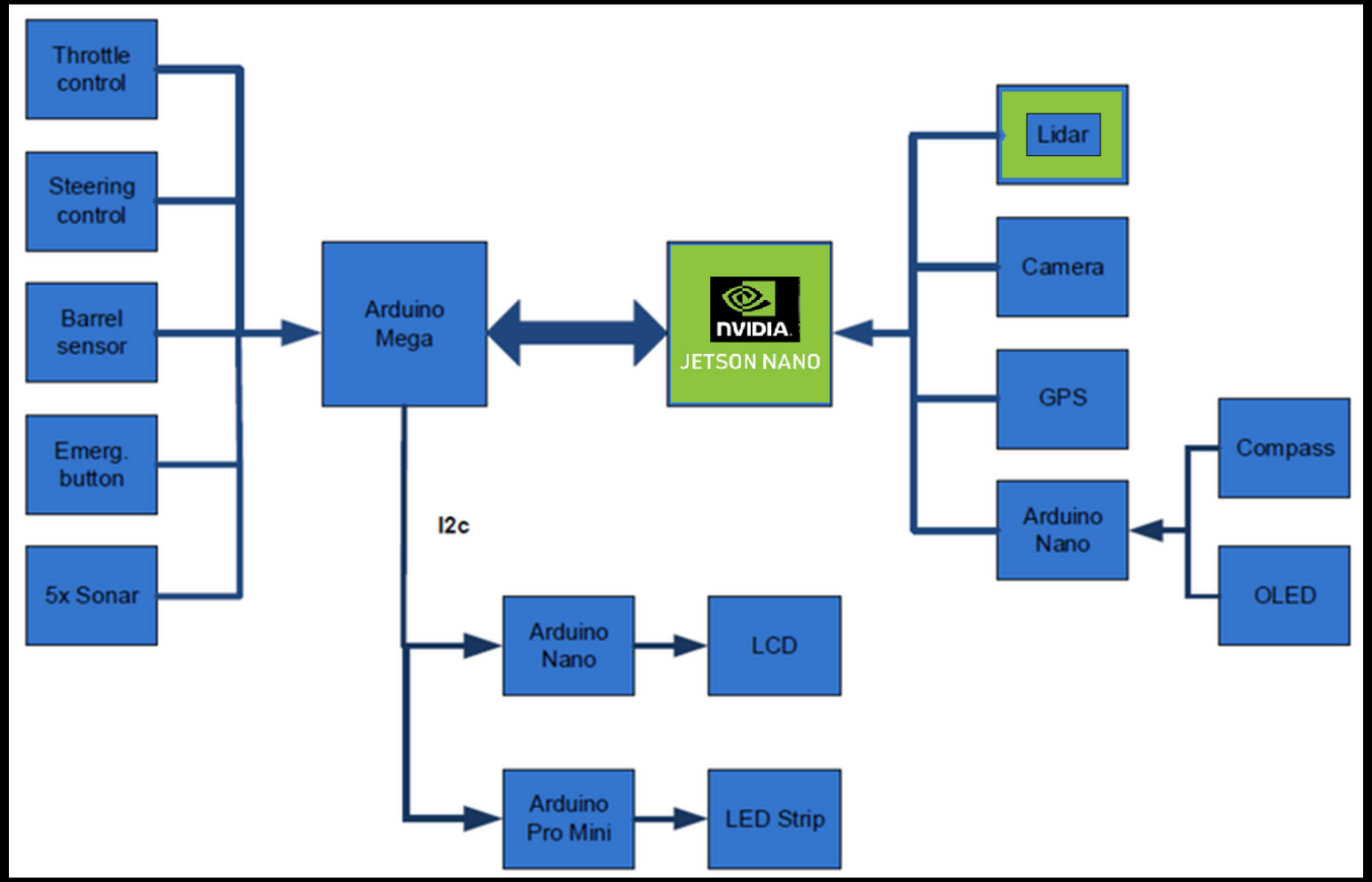
- [1] Adrián Matejov, Efficient Convolutional Neural Networks Recognizing Driveable Trails, Master's Thesis 2020
  - https://github.com/Adman/master-thesis/blob/master/efficient-cnns-recognizing-driveable-trails.pdf
- [2] Alasdair Allan: Benchmarking Edge Computing: Comparing Google, Intel, and NVIDIA accelerator hardware
  - https://medium.com/@aallan/benchmarking-edge-computing-ce3f13942245
- [3] Matt Cooper: When Conventional Wisdom Fails: Revisiting Data Augmentation for Self-Driving Cars
  - https://towardsdatascience.com/when-conventional-wisdom-fails-revisiting-data-augmentation-for-self-driving-cars-4831998c5509

THANK YOU

GENERAL SLIDES



HARDWARE DESIGN

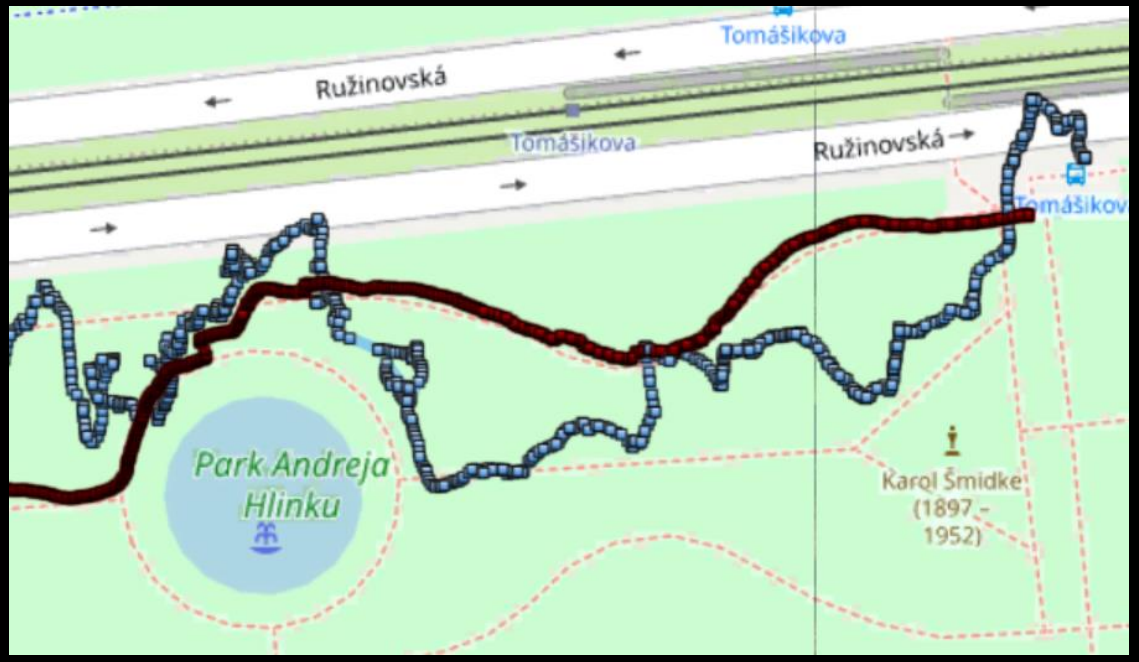


Jetson Nano vs Odroid-XU4

	Raspberry Pi3	Odroid-XU4	Jetson Nano
CPU	ARM Cortex-A53	Samsung Exynos5422 Cortex	ARM Cortex-A57
Clock	1.2 GHz	2 GHz	1,43 Ghz
Cores	4x	8x	4x
RAM	1GB LPDDR2	2GB LPDDR3	4GB LPDDR4
Flash	microSD	eMMC5.0 HS400	microSDXC Class 10 UHS-I
Ethernet	10/100 Mbit	1 Gigabit	1 Gigabit
USB	4× USB 2.0	2x USB 3.0 1x USB 2.0	4x USB 3.0

	R-Pi3	O-XU4	J-Nano
Image processing	167,3 ms	39,4 ms	27,3 ms
JPG/PNG writing	55 ms	17,7 ms	8,1 ms
Processing lag	2 sec	100 ms	0 ms (ramdisk)

GPS GROUND PLANE

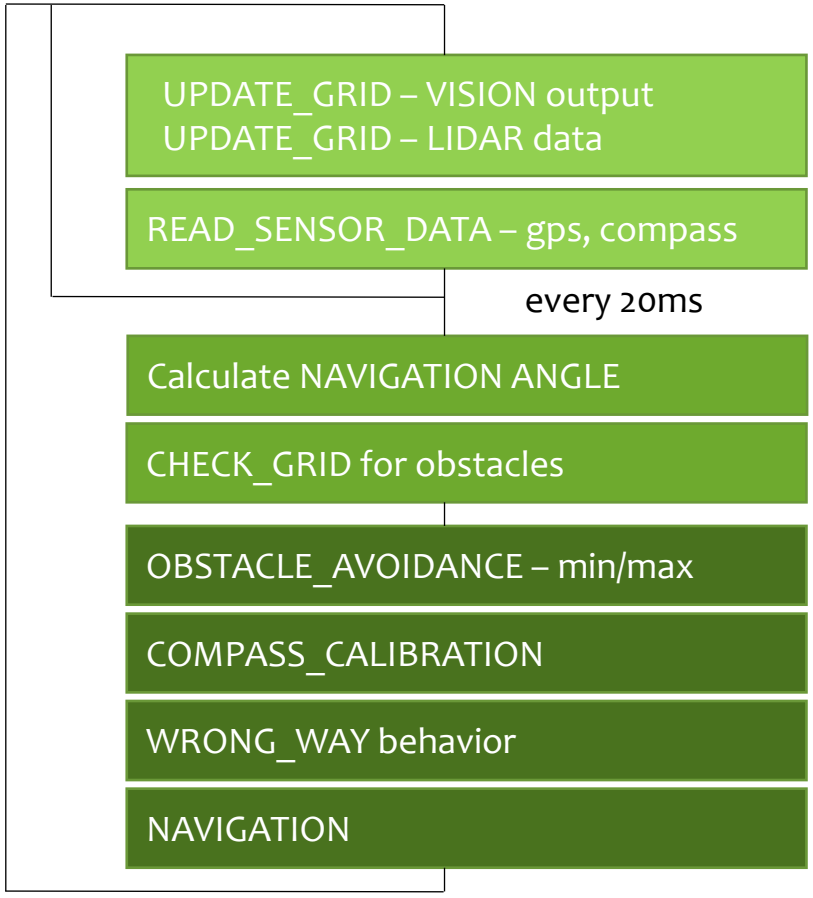


- U-Blox GPS Antenna documentation
  - Patch antennas - flat surface is ideal
  - can show very high gain, if mounted **on large ground plane (70x70mm)**
- USB 3.0 impact on GPS
  - Intel paper: USB 3.0* Radio Frequency Interference Impact on 2.4 GHz Wireless Devices
- We used simple shield for GPS
- Results: **great improvement (3m accuracy)**

170FOV CAMERA

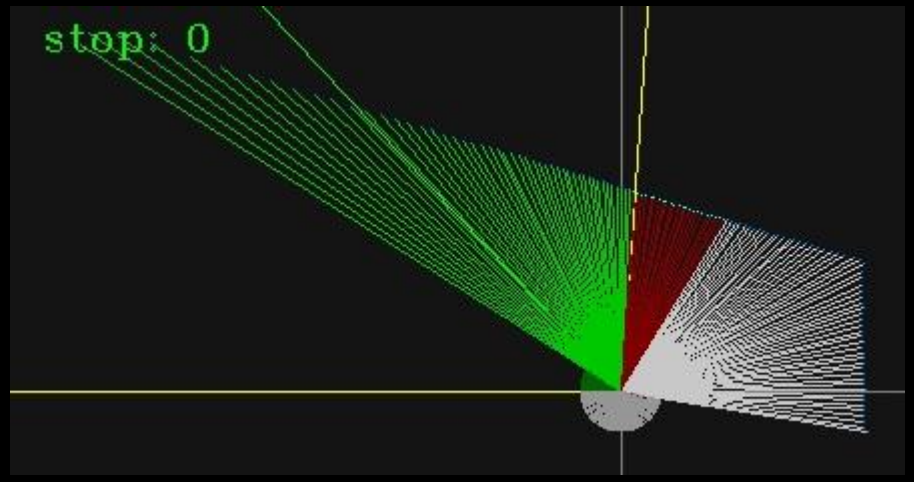
- **oCam : 5MP USB 3.0 Camera**
  - OmniVision OV5640 CMOS image sensor
  - Original lens Field Of View: 65 Degree
- Exchangeable Standard M12 Lens
  - Separate: 170 Degree Wide Angle
  - (standard accessory also for GoPro cameras)
- we 3d-printed an adjustable camera holder (fixed by screws)
- we broke the USB connector during the first test drive

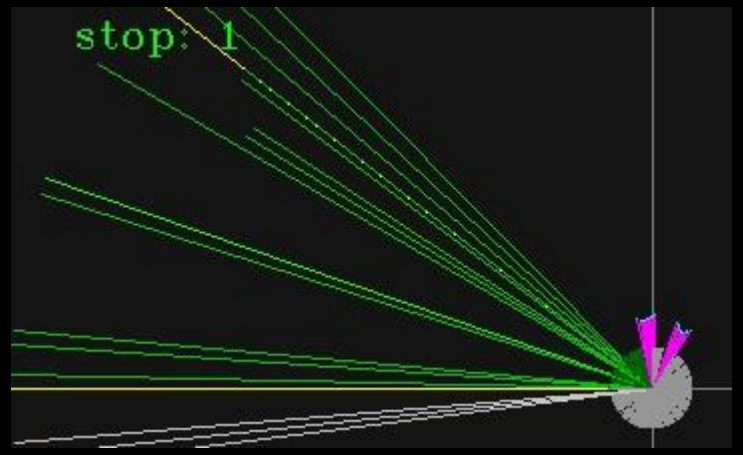
SOFTWARE DESIGN – PROCESSING



- LIDAR data capture – 15 Hz
- CAMERA capture – 30 fps
- VISION processing – 28 ms
- GPS data capture
- CONTROL BOARD comm.
- SAVE images to disk – 3x /sec

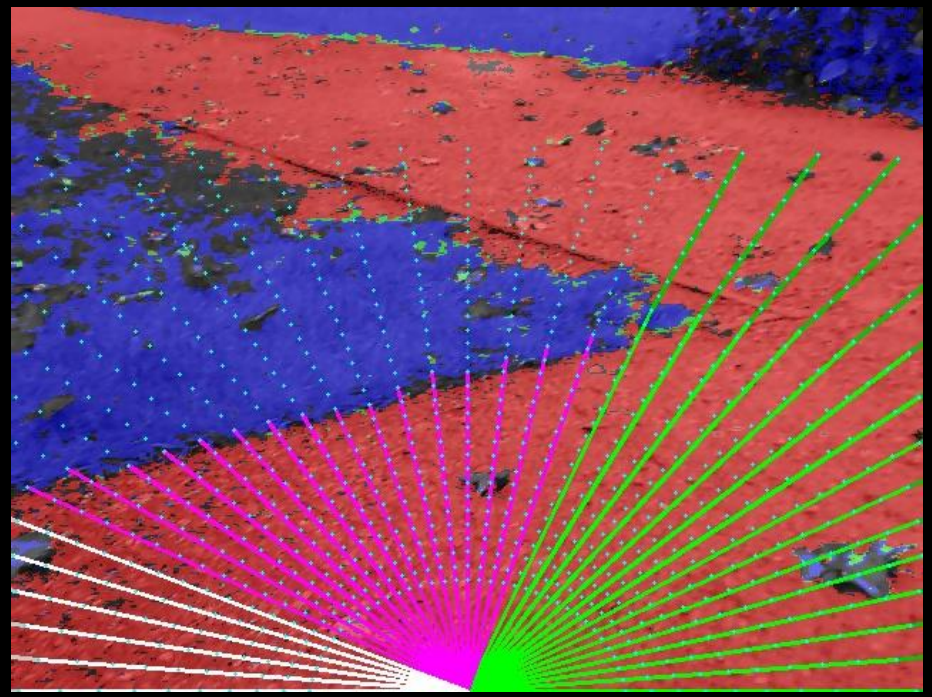
LIDAR – obstacle detection





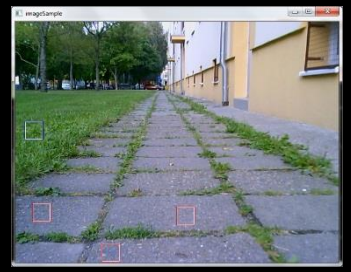
- Obstacle detection condition (red):
  - If distance is < 100 cm
  - Filtering: distance < 1cm (grey)
- Stop condition (pink):
  - Check angle: -45 to +45 degrees
  - If distance is < 50 cm at 3 diff. degs
  - Sonars were also used (rain issue)
- Obstacle avoidance (green/white)
  - Find OK intervals of > 20 degrees
  - Choose the closest to going straight

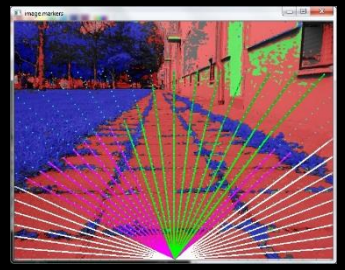
VISION – approach

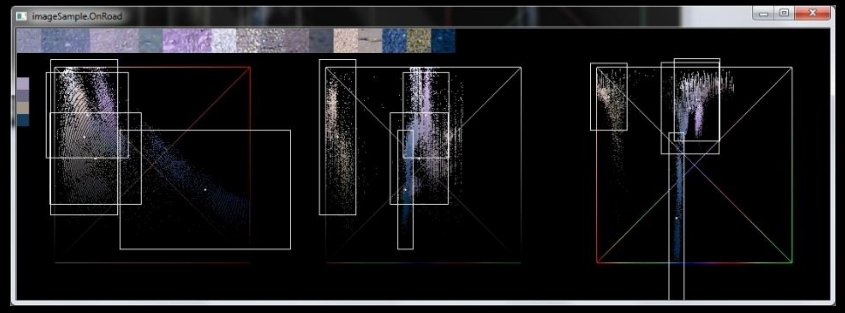


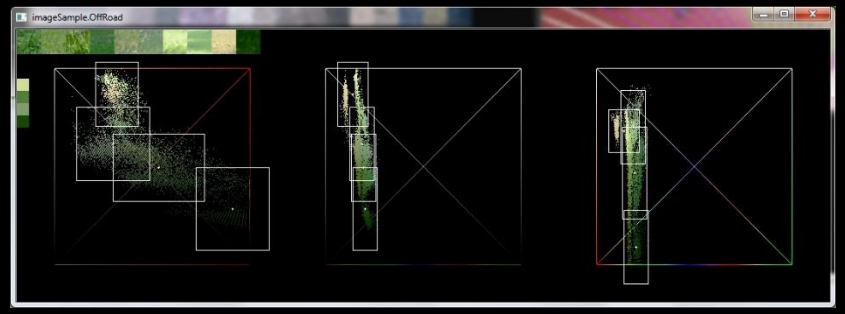
- Our approach: **lidar-like local map**
  - For any seen angle is obstacle closer than 1 meter?
  - 1 meter or to the image border
- Algorithm:
  - Pixel color classification
  - Evaluate grid points
  - Calculate distance to obstacle
  - Find OK intervals – same like LIDAR

VISION – Pixel color classification



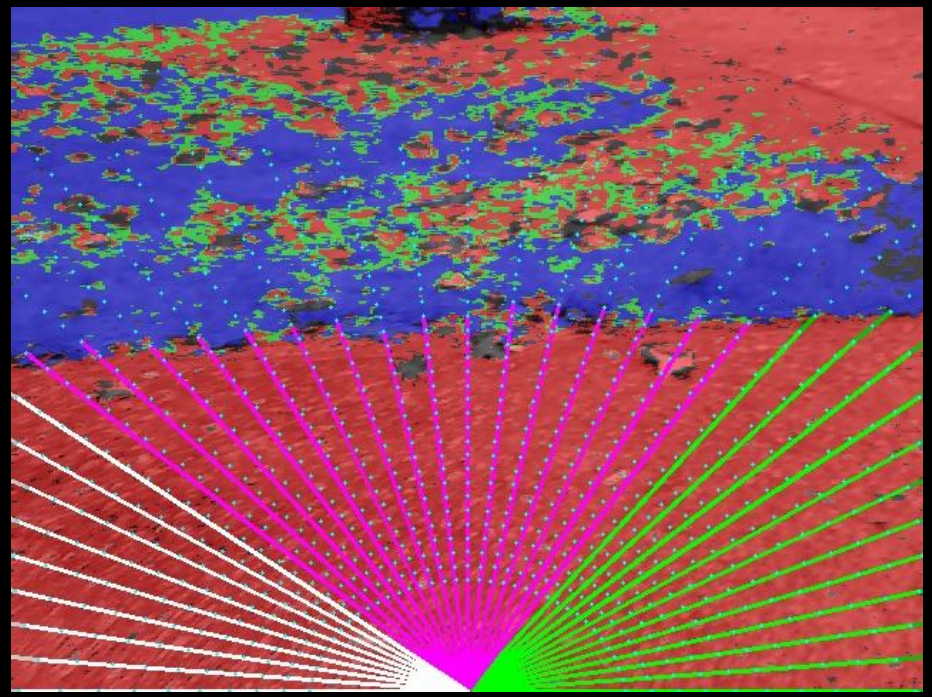






- Approach:
  - Choose sample pixel blocks (32x32) from training images
  - Calculate 4 clusters centers in color space (OpenCV kmeans)
  - Calculate cluster radius (histogram based)
  - Repeat for 2 classifiers : road and off-road (grass)
- HSV color space + Euclidian distance
- tool to evaluate images and to select sample blocks

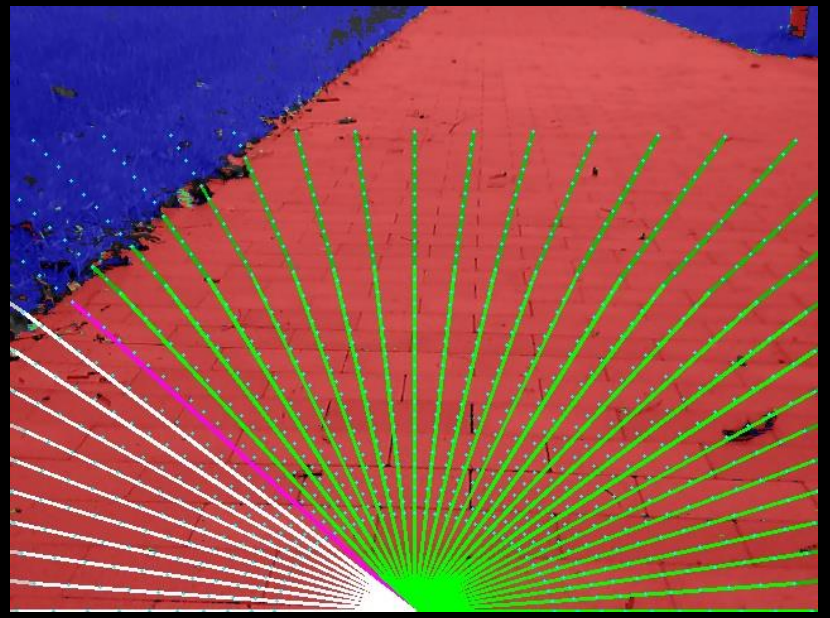
VISION – Algorithm

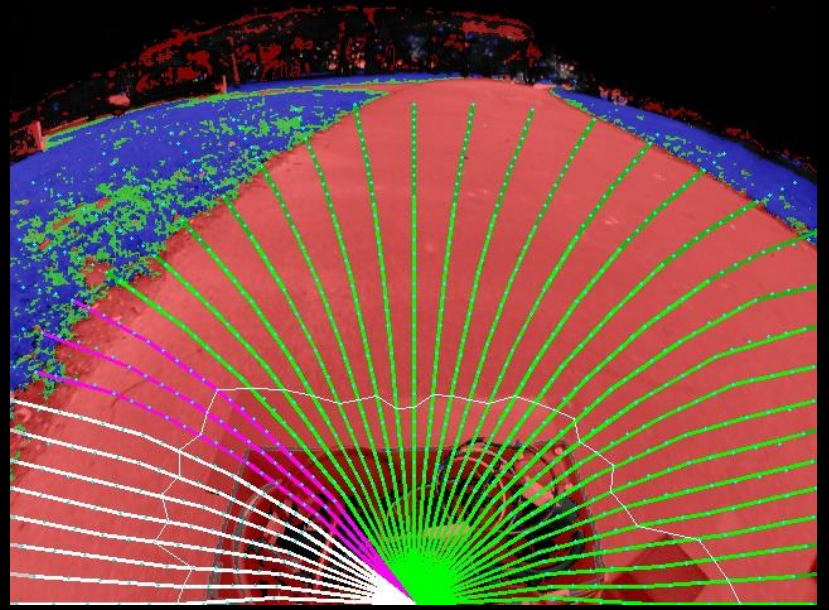


- Pixel color classification - 4 results:
  - Road (red)
  - Off-road (blue)
  - Both (green)
  - None (grey)
- Evaluate grid points
  - Cca 1000 points in 37 lines (5 deg)
  - Evaluating nearby pixels (80x80)
  - Majority of “Road” pixels is checked
- Calculate distance to obstacle
- Find OK intervals + merge with LIDAR

FISHEYE IMAGE PROCESSING

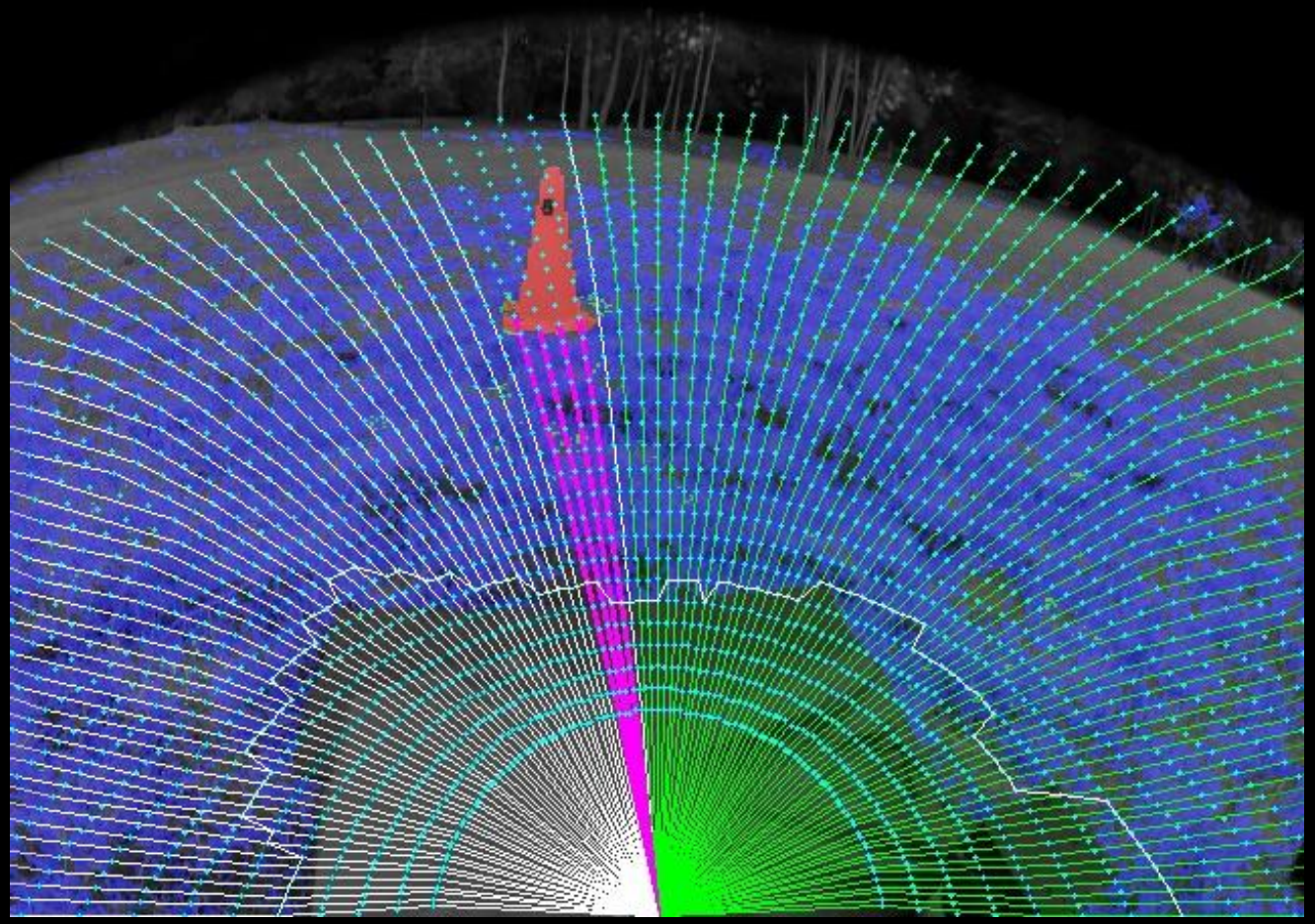
- **image transformation**
  - correction of mapping between XY points and local map coordinates – curved lines
  - OpenCV undistort() was not used (because of slow performance)
- **front side of the robot on every image**
  - was fixed by SW masking
  - presents an issue for training a neural network
- **sky was “masked” using a black tape**
  - for avoiding white ballancing camera issues





VISION - ROBOORIENTEERING

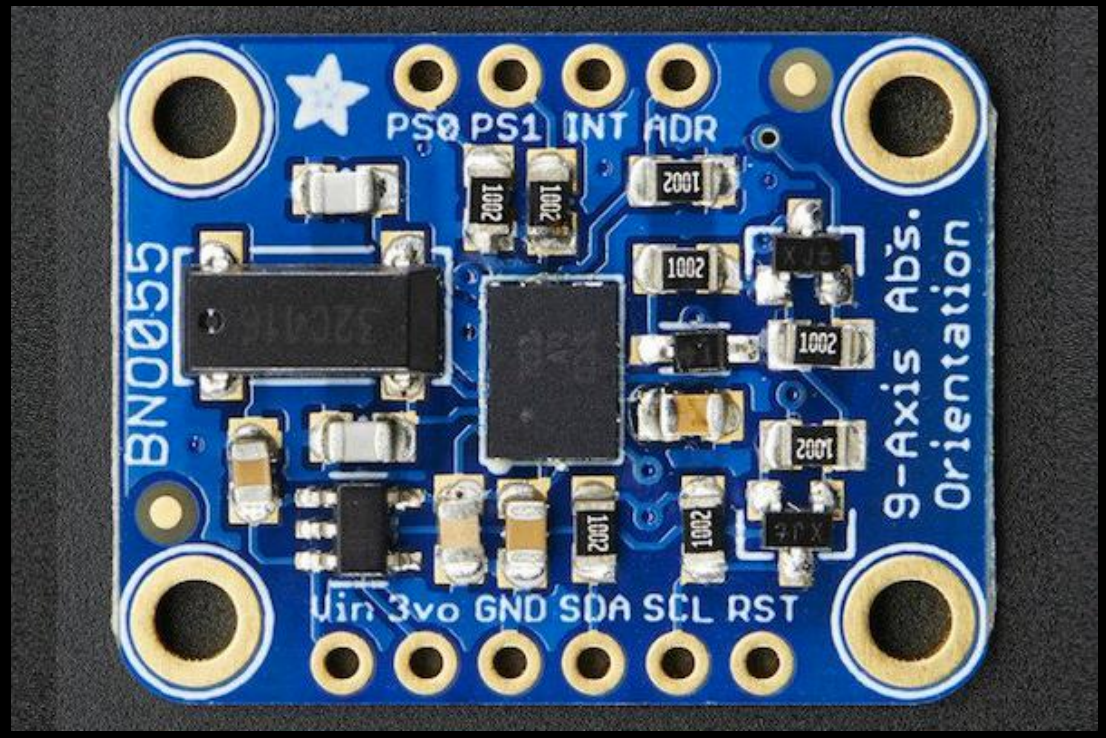
- Vision algorithm was modified to detect orange cones



NAVIGATION – ROUTE PLANNING

- OpenStreetMap data export
  - filter segments: footway, track with <grade3
- **Dijkstra's Shortest Path Algorithm**
  - 418 nodes, 504 segments
- Performance: < 2ms
- Visualisation: kml export, cpp export
- Navigation: keep azimuth towards a point that is 10m along planned route

COMPASS ISSUES

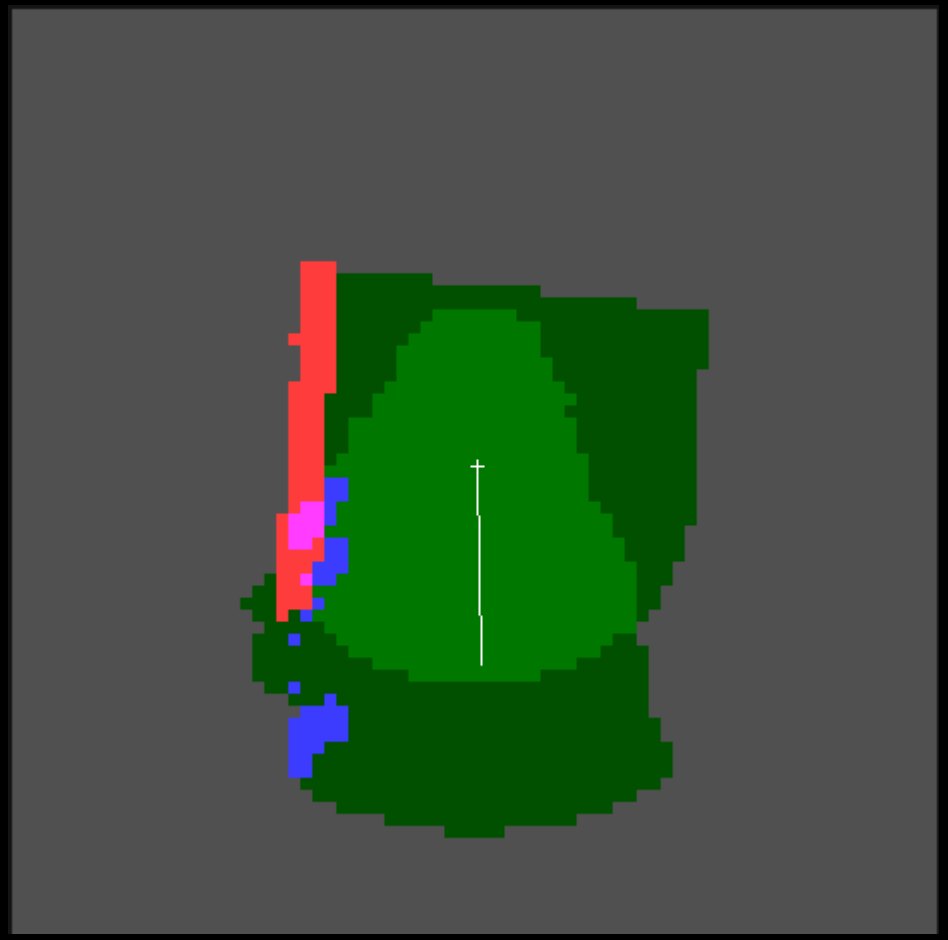


- Compass vs GPS calibration:
  - interval of 7 seconds - robot is moving straight
  - gps and compass azimuths to be fixed
  - could be performed every 30 seconds – 3 minutes
- Issues:
  - after Round #1 very inaccurate results – delta values between 45° and 90 °
  - robot is sometimes slightly left turning
  - local changes during the day (bridge)
- for testing in Bratislava and also for competition we used fixed value 60 °

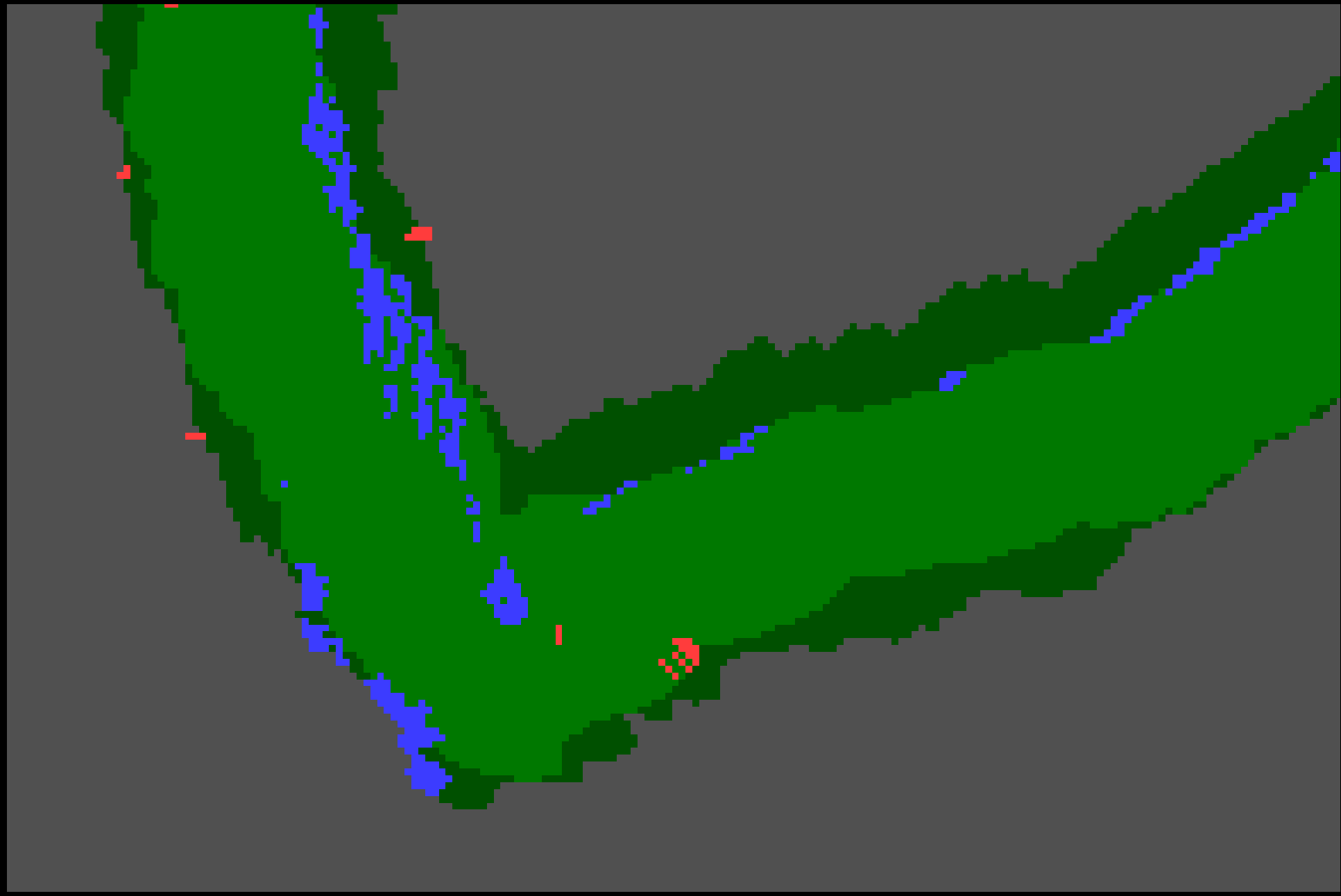
QR CODES

- **ZBar bar code reader**
  - open source software suite
  - supports: EAN-13/UPC-A, UPC-E, EAN-8, Code 128, Code 39, Interleaved 2 of 5 and QR Code
- Performance impact:
  - one execution: 50-200ms
  - our target 30-50ms per frame
- Solution: images are scanned for QR codes only when waiting for navigation coordinates

WORLD MAP – BUILDING LOCAL GRID



- Local grid map
  - to store polar information from lidar and vision
  - 1 cell: 10 x 10 cm, 1 byte pre cell, array[2000][2000]
  - always overwriting with new data - no heatmap
- Local position taken from GPS + odometry
  - wheel encoders provide speed information
- Colors used for visualization
  - Blue – grass (not-a-road) detected
  - Red – lidar obstacle
  - Green - no obstacle (light green = both sensors)



VISUALISATION

- Visualization in web browser
  - works on notebook/tablet/phone
- Messages from log files
  - last set of lines matching selected substrings
  - Info: robot display, gps, processing, calibration
- Log parser is exporting interesting data do .json file
- Web page performs Ajax JSON requests every 1 sec
  - Images are only downloaded on demand (checkbox)