ROS Framework used by Team Hector Darmstadt

ROS Workshop Koblenz 2011
Stefan Kohlbrecher, Karen Petersen, Thorsten Graber, Johannes Meyer
Outline

- Introduction
- Hardware Platforms
- Building Blocks for a (Semi-)Autonomous Rescue Robot
  - System Overview
  - Drivers and Controllers
  - Localization and Mapping
  - Navigation and Path Planning
  - Victim/Object Detection
  - High-Level Control
  - Human-Robot Interaction
  - Simulation

- Current state of ROS packages
Team Hector is part of the RTG1362: “Cooperative, Adaptive and Responsive Monitoring in Mixed Mode Environments”
Example
Monitoring in Normal Operation
Example
Some Monitoring Elements and Channels Knocked-Out
Motivation
Deployment of Additional Equipment (Robots, Sensors)

→ Motivates fundamental research questions being addressed by our RTG
Team Hector

- Hector: Heterogeneous Cooperating Team of Robots

- Established in Fall 2009

- 9 PhD Students involved in the past:
  - Mykhaylo Andriluka, Martin Friedmann, Johannes Meyer, Stefan Kohlbrecher (team captain), Karen Petersen, Christian Reinl, Paul Schnitzspan, Armin Strobel, Thorsten Graber

- Transition from RoboFrame to ROS as the central middleware since late 2010
Team Hector


3rd place (out of 12 & 16 teams) at the SICK Company’s Robot Day, October 2009 & 2010, Waldkirch

2nd place (out of 27) “Best in Class Autonomy” at RoboCup 2010, Singapore

Winner and “Best in Class Autonomy” at Robocup 2011 GermanOpen

2nd place (out of 27) “Best in Class Autonomy” at RoboCup 2011, Istanbul
Hardware Platforms
Hector UGV

Pan/Tilt Camera Head
- Daylight Camera
- Thermal Camera
- RGB-D Camera

Vision Computer
- Core 2 Duo CPU
- Nvidia GPU

2 actuated LIDAR sensors

R/C Car Chassis
- 4-Wheel-Steering
- 1:5 Gear
- Wheel Encoders

Motion and Navigation
- μC Board / Geode LX
- IMU / GPS / Compass

Total HW Cost: approx. 14.000 Euro (including payload)
Endurance: 30 - 40 minutes
State-of-the-art computing power of a mobile PC onboard
Hardware Platforms
Hector Lightweight UGV

- Based on commercial platform “Wild Thumper”
  - Low cost (~300 EUR)
  - Good mobility
  - First tested at RoboCup 2011
- ROS Integration
  - Arduino based motorcontroller
  - fitPC2 (Atom Z530 1.6 GHz)
- Investigate low cost, low weight platform
- Ongoing work
  - Sensor arm integration using ROS arm_navigation
Building Blocks for a (Semi-)Autonomous Rescue Robot

- **Requirements**
  - Robust and flexible hardware platform
  - Autonomous exploration of unknown, complex environments
  - Detection of victims and other objects of interest
  - Interaction with other robots/with human rescue forces/with a human supervisor

- **Some problems to be solved…**
  - Self localization including full 3D pose estimation
  - Environment perception and mapping (2D or 3D)
  - Path planning and control
  - Detection, identification and tracking of objects using multiple cues
  - Reliable communication infrastructure
  - High-level decision making based on all inputs and external communication
System Overview

- Navigation
- Localization and Mapping
- Perception
- High-level Control
- Drivers
Hardware Drivers and Controllers

- **Real-time driver** running in Xenomai-enabled Linux
  - Speed control based on wheel odometry
  - Servo control
  - Interface to the integrated sensors

- **Controller** nodes provides platform independent motion interface
  - Drive to point, follow path
  - Look at point (using tf)

- **Pose Estimation** node
  - Extended Kalman Filter estimating the fused 9-DOF state vector (orientation, position, velocity)
Localization and Mapping

The SLAM problem

- Build a map representation of the environment and simultaneously localize the robot within that map:
  - Range sensors / Laser scanner (LIDAR)
  - (3D-/Stereo-) Camera

State of the Art:

- 2D SLAM: Gmapping, …
  - planar environments, odometry
- 3D SLAM: SLAM6D, …
  - currently not applicable for online use
- Visual SLAM: SBA, RGB-D-SLAM…
  - computationally expensive and error prone
Localization and Mapping
Occupancy Grid Mapping

- Map is represented by a 2D grid holding the probability $P_{xy}$ of cell occupancy

1. Scan Transformation
   - Transformation of laser rays into the map frame

2. Scan Matching
   - Alignment of incoming laser scans to the **map**

3. Map Update
   - Increase $P$ for each ray endpoint
   - Decrease $P$ for free cells

➢ **Efficient map query!**
Localization and Mapping
Inertial Navigation System

- Estimation of the **full 3D state** (position, orientation, velocity) of the robot from different sensor sources:
  - Inertial Measurement Unit (IMU)
  - Compass (Magnetic Field)
  - Global Satellite Navigation
  - Altimeter, Range Sensors etc.

Problems:
- Absolute position is not very accurate or not available at all
- Solution suffers from drift
- Acceleration can lead to significant orientation errors
Localization and Mapping
Navigation Filter

- Sensor information is fused using an **Extended Kalman Filter (EKF)**

- **State Vector**
  \[
  \bar{x}_k = (\Omega_k^T, p_k^T, v_k^T, \delta \omega_k^T, \delta a_k^T)^T
  \]

  - Orientation
  - Position
  - Velocity
  - Angular rate error
  - Acceleration error

- **System Input** (= Inertial Measurements)
  \[
  u_k = (\omega_k^T, a_k^T)^T
  \]

  - Angular rates
  - Accelerations
Our approach: Couple both localization approaches in a loose manner.
Pose Update from SLAM:

- Pose estimates from EKF and SLAM have unknown correlation!
- Solution: **Covariance Intersection (CI)** approach [5]

\[
(P^+)^{-1} = (1 - \omega) \cdot P^{-1} + \omega \cdot C^T R^{-1} C
\]

\[
\mu^+ = P^+ \left( (1 - \omega) \cdot P^{-1} \mu + \omega \cdot C^T R^{-1} z \right)^{-1}
\]

with

- Estimated state and covariance (a-priori): \((\mu, P)\)
- Scan Matcher pose and covariance: \((z, R)\)
- Observation matrix \(C\)
- Tuning parameter \(\omega \in [0, 1]\)
Localization and Mapping
Trajectory Server

- Logs Trajectory based on tf data
- Make Data available as nav_msgs::path via
  - Regularly published topic
  - Service
- Currently used for
  - Visualization
  - GeoTiff node
- Can log any tf based trajectories
Localization and Mapping
GeoTiff node

- Provides RC Rescue League compliant GeoTiff maps
- Trigger for saving the map
  - Regular Intervals
  - On Request (topic)
- Runs completely onboard
- Uses ROS services to retrieve
  - Map
  - Travelled path
  - Victim Locations
Integration of our SLAM system in a small hand-held device

- Intel Atom processor
- Same hardware as on our quadrotor UAV
- Optional connections to GPS receiver, Magnetometer, Barometer for airborne application
Localization and Mapping
Handheld Mapping

- RoboCup 2011 Handheld Mapping System dataset
- Small Box with
  - Hokuyo UTM-30LX LIDAR
  - Low Cost (<100$) IMU
  - Atom Z530 1.6 GHz board
- Available at our GoogleCode repository
Localization and Mapping
USV mapping

- SLAM System mounted on USV (Unmanned Surface Vehicle)
- Self-Contained, no interconnection with USV (apart from power supply)
Localization and Mapping
ROS Graph (1)

- hector_pose_estimation
  - /poseupdate
- hector_mapping
  - /tf
  - /map
- hector_map_server
  - /get_distance_to_obstacle
  - /map
- hector_trajectory_server
  - /trajectory
- hector_geotiff
  - /objects
  - /save_geotiff
- Nodes
  - Topics
  - Services
Motivation:
- Elevation map is mandatory for ground robots
- For detection of
  - stairs
  - ramps
  - step fields
  - ...
- No generic ROS package!

Proposed Map Representation:
- Two-dimensional array \((x,y)\)
- Height value \((h)\)
- Variance \((\sigma^2)\)
Localization and Mapping
Elevation Mapping

Kalman Filter based Approach:

\[
 h(t) = \frac{1}{\sigma_z^2(t) + \sigma_{h(t-1)}^2} \mathbf{G}^2 z(t) h(t-1) + \sigma_{h(t-1)}^2 m(t) \\
 \sigma^2_{h(t)} = \frac{\sigma^2_z(t) \sigma^2_{h(t-1)}}{\sigma^2_z(t) + \sigma^2_{h(t-1)}
\]

More precisely:

\[
 h(t) = \begin{cases} 
 z(t) \\
 h(t-1) \\
 \frac{1}{\sigma^2_m + \sigma^2_{h(t-1)}} \mathbf{G}^2 m h(t-1) + \sigma^2_{h(t-1)} m(t)
\end{cases}
\]

\[
 \sigma^2_{h(t)} = \begin{cases} 
 \sigma^2_z(t) & \text{if } z(t) > h(t-1) \land dm < c \\
 \sigma^2_{h(t-1)} & \text{if } z(t) < h(t-1) \land dm < c \\
 \frac{\sigma^2_z(t) \sigma^2_{h(t-1)}}{\sigma^2_z(t) + \sigma^2_{h(t-1)}} & \text{else}
\end{cases}
\]

where \( dm \) denotes the Mahalanobis distance:

\[
 dm = \frac{\mathbf{G}(t) - h(t-1)^2}{\sigma^2_{h(t)}}
\]

Reference:
Localization and Mapping
ROS Graph (2)
Navigation and Path Planning
Cost Mapping

Motivation:
- Avoid planning a path over untraversable regions
- Integration of depth and LIDAR information

Proposed Map Representation:
- Two-dimensional array (x,y)
- Occupancy grid map
  - Unknown
  - Free
  - Occupied

\[ \text{Elevation Map} \rightarrow \text{Elevation Cost Map} \]

\[ \text{SLAM Map} + \]

\[ = \]

\[ \text{Merged Cost Map} \]
Laplacian-of-Gaussian Filter kernel based Approach:

1. **Elevation Map**
   - Apply **Low-Pass Filter**
     - Remove all traversable steps
2. **Smoothed Map**
3. **Elevation Cost Map**
   - Apply **High-Pass Filter**
     - Highlight all the rest

**Elevation Map**

**Laplacian-of-Gaussian Filter**

**Elevation Cost Map**
Navigation and Path Planning

Exploration

- Exploration Transform / Frontier based
- Uses CostMap2D based cost map
- Plugin for move_base

Capabilities

- Generate target pose and path simultaneously (exploration)
- Plan path to given target pose
- Create frontier based target pose
Navigation and Path Planning

Path Planning

- Problem: Hector UGV is a non-holonomic vehicle
  - Navigation stack supports only
    - Holonomic
    - Differential drive
- Solution:
  - Use OMPL/SBPL Lattice Planner
  - Modify move_base
    - Fixed Cost Map
    - Low Level Trajectory Follower for SBPL paths
    - Repeated Execution of SBPL (dynamic replanning)
Navigation and Path Planning
ROS graph

- **/elevation_map**
- **/map**
- **hector_cost_map**
- **/cost_map**
- **costmap_2d_ros**
- **hector_move_base**
- **/move_base/goal**
- **/move_base/cancel**
- **hector_global_planner**
- **hector_exploration**
- **hector_local_planner**
- **/cmd_vel**
- **/drivepath**
- **monstertruck_controller**

**Nodes**
- Plugin
- Topics
- Services
Victim and Object Detection
Thermal Victim Detection

- Search for groups of connected pixels with temperature of human body
- Significantly less reliable than visual people detection (in real-world scenarios)
- Define confidence $s^\text{therm} \in [0, 1]$ proportional to the number of pixels within human body temperature range
Victim and Object Detection
Visual Victim Detection

- Detection of upper bodies in camera images based on
  - Histograms of Oriented Gradients (HOG)
  - Discriminative SVM classifier
- Parallel computation on GPU
- Define hypothesis confidence $s_{vis} \in [0, 1]$ derived from SVM score
- Generalization to other kind of objects
Victim and Object Detection
Object Association and Tracking

- Tracking of uncertain victim/object estimates $\{x^j_k, P^j_k, \pi^j_k\}$ over time
  - $x^j_k, P^j_k$: 3D position vector and covariance
  - $\pi^j_k$: Confidence ($1 - \text{error probability}$)

- Robot position and camera transformation are assumed to be known

Algorithm:

- For each new hypothesis $\{z_k, s_k\}$ with $z_k = \begin{bmatrix} d_k, \alpha_k, \beta_k \end{bmatrix}^T$:
  - Data association: Find best matching victim estimate $j^*$ that minimizes a distance measure in measurement space (or instantiate a new one)
  - Position update: Update position using an EKF that additionally considers the hypothesis confidence $s_k$
  - Confidence update: Increase victim confidence according to
    \[ \pi^j_k = \pi^j_{k-1} + s_k \cdot (1 - \pi^j_{k-1}) \]
  - Negative update: Decrease confidence of all estimates not being observed despite estimated position is within FOV (optional)
Victim and Object Detection
Hazmat Signs

- Detection of signs of hazardous materials
- HOG cannot distinguish between different objects with similar shape
- Augmentation through color histograms in LAB-space for classification
- Two step approach:
  - Search for candidates with HOG / SVM
  - K-nearest-neighbor classification
Victim and Object Detection
Victim Depth Verification

Motivation:
- Verify thermal and visual victim hypotheses
  - Eliminate false-positives
- Uses MS Kinect
- First experiments at Robocup 2011, Istanbul

Low-Level Hypotheses:
- A valid victim is measurable by a depth camera (point cloud)
- A valid victim is not flat

Verification Request \((x, y, z)\):
- Check if there are enough valid depth points at \((x, y, z)\)
- Check if all depth points lie in a plane
  - no
  - yes

Victim verified
Victim cancelled
Victim and Object Detection
ROS Graph

SearchAndRescueApp (RoboFrame)

 chest/image

object_tracker

verification_services [0..*]

/object_update /objects

add_object

/set_object_state

get_distance_to_obstacle

/object_percept /pose_percept

/camera/image /thermal/image

/camera/depth/points

victim_verification

/verify_victim

Topics

Services

Nodes
High-Level Control

Task Allocation

- Modules generate tasks for desired actions
  - Explore area
  - Verify victim hypotheses
- Calculate cost to execute tasks
  - Metric can include
    - Estimated duration to accomplish a task
    - Assumed gain of task execution
    - Expected quality of solution
- Scale with task priority

Task Allocation
- Single robot: Greedily allocate task with lowest cost
- Multiple robots: Use more sophisticated task allocation algorithm, for example market-based
- Allow re-allocation if high-priority tasks emerge
High-Level Control
Behavior with XABSL

- Details on how to execute specific tasks
  - Request desired position + orientation of robot
  - Move camera

- Realized with XABSL (eXtensible Agent Behavior Specification Language)
  - Hierarchical finite state automata
  - Each state machine is called option
  - Each state can execute other options or low-level behaviors
  - Intermediate code is interpreted by the XABSL-Engine

- Reusability of fine-tuned behavior in many high-level options
- Behavior can be modified during runtime
Human-Robot Interaction

- Teleoperation
  - Direct control of robot motions
  - Control of cameras
- Semi-autonomous operation
  - Goal selection via point-and-click
  - Autonomous path planning
- Supervision of robot teams
  - Control team coordination
  - Modify mission details
  - Influence task allocation

- GUI so far based on RoboFrame
  → New solution will be presented after lunch
Simulation

- Based on the Multi-Robot-Simulation-Framework MuRoSimF
  - Software-in-the-loop testing
    - Run same code as on real robots
    - Test software functionality without hardware
  - Arbitrary mazes and robots
  - Simulation of various sensors
    - Camera
    - Thermal camera
    - Laser range finder
  - Ground-truth data
    - Simplifies evaluation and tuning of algorithms
  - Communication between ROS and robot firmware
Some Additional Considerations

- **/syscommand**
  - Reset internal state of all nodes simultaneously

- **tf package**
  - No clear definition of frames
Open Questions

- **Simulation environment** providing a complete Rescue arena
  - Gazebo
  - USARsim
  - Morse
- **Multi-robot scenarios**
  - Multi-master
  - Map merging
- **Integrated GUI solution**
  - RViz + Tools
  - General Rescue GUI
# Current State of ROS packages

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>monstertruck_driver</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>hector_pose_estimation</td>
<td>✓ (✓)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>hector_mapping</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>hector_trajectory_server</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>hector_geotiff_node</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>hector_elevation_mapping</td>
<td>●</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>hector_move_base (+ Plugins)</td>
<td>●</td>
<td>●</td>
<td>✗</td>
</tr>
<tr>
<td>object_detection</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>object_tracking</td>
<td>●</td>
<td>✓</td>
<td>●</td>
</tr>
<tr>
<td>victim_depth_verification</td>
<td>●</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>SearchAndRescueApp</td>
<td>✗ (Roboframe)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>SearchAndRescueGui</td>
<td>✗ (Roboframe)</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>