TAS Project: Introduction to ROS

Laith Alkurdi
`laith.alkurdi@tum.de`

Institute of Automatic Control Engineering
Technische Universität München

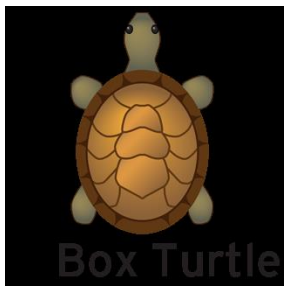
An introduction to ROS

- Items covered today:
  1. What is ROS?
  2. How to Install ROS?
  3. The ROS filesystem
  4. ROS graph concepts
  5. Basic commands in the command line
  6. ROS on mobile platforms: SLAM, AMCL, Navigation stack
  7. Key concepts, features

What is ROS?

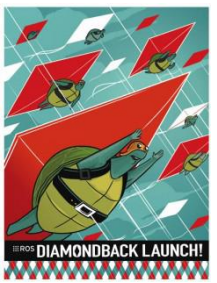
- The Robotic Operating System is a software framework for robot software development. It provides a functionality similar to that of an operating system on a heterogeneous computer cluster.
- Started in 2007 at the Stanford AI lab and was called **switchyard.**
- Now at the research institute Willow Garage.
- Version updates over time:

Hydro
4.09.13

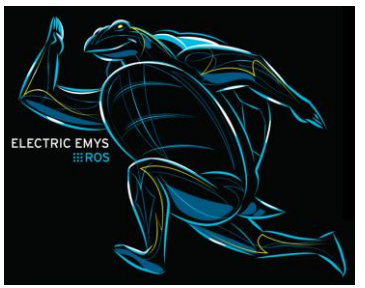


Box Turtle
02.03.10

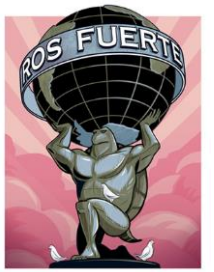
C Turtle
02.08.10



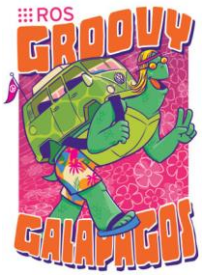
Diamondback
02.03.11



Electric
30.08.11



Fuerte
23.05.12



Groovy
31.12.12

ROS-Features

- ROS resembles a real operating system (but not really is one)
  - It has hardware abstraction
  - It is independent of programming language
  - Already has a wide range of commonly used algorithms implemented
  - It has a powerful message passing system between processes. This message passing system is OS-independent.
  - It has a very powerful standard packaging management system. Community driven projects.
  - Useful set of shell commands with TAB completion.

ROS as a distributed communication system

- ROS is structured as a peer-to-peer network of nodes (processes, that does something!) that are loosely coupled at runtime, they share messages using:
  - Service: synchronous style of communication
  - Topics: asynchronous data streaming communication
  - Parameter server: data storage

What can I use to program in ROS

- As we mentioned already, ROS is language independent,
  - Python
  - C++
  - Lisp
  - Java (experimental)
  - Lua

What about operating systems?

- The supported operating system is Ubuntu
  - For this work you must download 14.04 LTS+ROS Indigo

- There are other experimental operating systems:
  - Windows
  - Mac OS X
  - Debian
  - Fedora
  - Gentoo
  - Arch
  - OpenSuse
  - Angstrom
  - Few others.

What are the supported robots?

- Husky, which is a rugged outdoor-ready unmanned ground vehicle (UGV) is fully supported by ROS and all the packages that has been developed for it is available on github! Also with simulator file in the husky_simulator package.
- The NAO robot and willow garage's PR2 use ROS
- Lots of robots!

Actually …

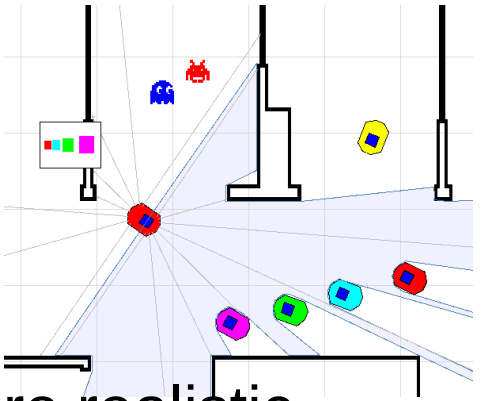
- Lets look at a couple of videos …

ROS also supports different sensors too…

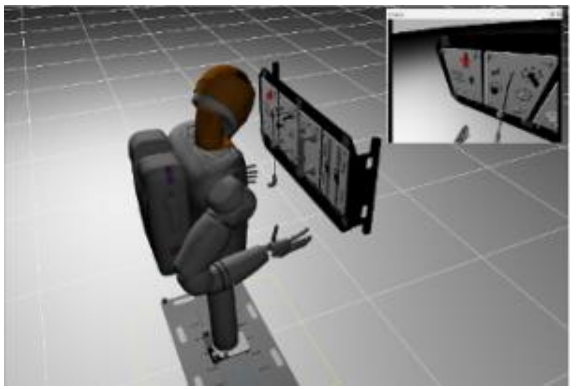
- Basically the who idea is we get a robotic operating system where robotic engineers do not have to reinvent the wheel every time they need a specific method/sensor driver/ robot … etc
- 1D/2D/3D range finders
- Cameras
- Force/torque/touch sensors
- Motion Capture systems
- Pose estimation (IMU/GPS)
- Audio/ Speech recognition
- RFID
- Sensor/ actuator interfaces

ROS supports simulators too..

- Stage is a 2D simulator. It can support multiple mobile robots.
  - Has sensor/actuator models
  - Has models for manipulation
  - Doesn't contain a physical model, so can not model collisions etc



- Gazebo is a 3D simulator of multiple robots in a more realistic setting.
  - Has a physics model
  - Supports complex robots/actuators/sensors
- Webots
  - Also a very powerful simulator



How can we install ROS?

- The best way to install (actually learn about) ROS is following the instructions in http://wiki.ros.org/ROS/Installation/ubuntu
- Configure your Ubuntu repositories
  - Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse."
- Setup sources.list
  - sudo sh -c `echo \deb http://packages.ros.org/ros/ubuntu trusty main" > /etc/apt/sources.list.d/ros-latest.list‘
- Setup keys
  - wget http://packages.ros.org/ros.key -O - j sudo apt-key add –
- Install ROS Desktop-Full,
  - sudo apt-get install ros-indigo-desktop-full

Installing ROS

- Environment setup
  - echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
  - source ~/.bashrc
- Get ROS install
  - sudo apt-get install python-rosinstall

- Once you are done start with the tutorials! Best way to learn ROS..
  - http://wiki.ros.org/ROS/Tutorials

ROS file system

- We have two major levels: Metapackages, and then a Package
- At the lowest level of software organization in ROS we have what is called a **Package**
  - Has one purpose: read sensors/write to motors etc
  - Comes with what is called a **Manifest** *manifest.xml*
  - A Manifest is a description of a package and what other packages it depends on
- On a higher level we have what is called a **Metapacakages(stacks)**
  - A Metapackage is a number of **Packages**

- Navigating between packages and nodes can be complicated using only **ls**, **cd** .. ROS provides some helpful file system tools

What kind of commands can we use in bash for ROS

- rospack
  - List : Print newline-separated list for all packages
  - Find : return the absolute path to a package
  - Depends : return a list of all of a package's dependencies
  - Profile :Force a full crawl of package directories
- roscd
  - Roscd turtlesim
- rosls
  - Rosls turtlesim
- rosed
  - Echo $EDITOR/ export EDITOR=gedit
  - Rosed turtlesim manifest.xml
- roscd

Lets make a package!

- Go over the catkin tutorials in ROS tutorials webpage..
- Make a catkin package:
  - $ mkdir -p ~/catkin_ws/src
  - $ cd ~/catkin_ws/src
  - $ catkin_init_workspace

packages

- Roscd sandbox
  - Roscreate-pkg sample std_msgs rospy roscpp
  - Rospack Locate, depends
  - Rosed sample manifest.xml
- Rosmake sample

What is a ROS node?

- A node is an executable that uses ROS to communicate with other nodes.
- A node really isn't much more than an executable file within a ROS package.
- ROS nodes use a ROS client library to communicate with other nodes.
- Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service.
- Nodes can be:
  - Controlling a motor (velocity, steering angle)
  - Do SLAM, AMCL
  - Do path planning
  - Point cloud processing …

Roscore

- Is the first thing you should do before you run any of your nodes
- Roscore is the core node
- Stores topics and services registration information for ros nodes
- Nodes then establish connections as appropriate
- Allows nodes to dynamically create connections as new nodes are run

Messages

- Simply, they data structures consisting of typed fields
- There are standard primitive types which as supported
  - int{8-64}
  - Float{32-64}
  - String
  - Time
  - Duration
  - Array[]
- Nodes communicate with each other by passing messages
- The direction of the message is routed depending if its publish/subscribe

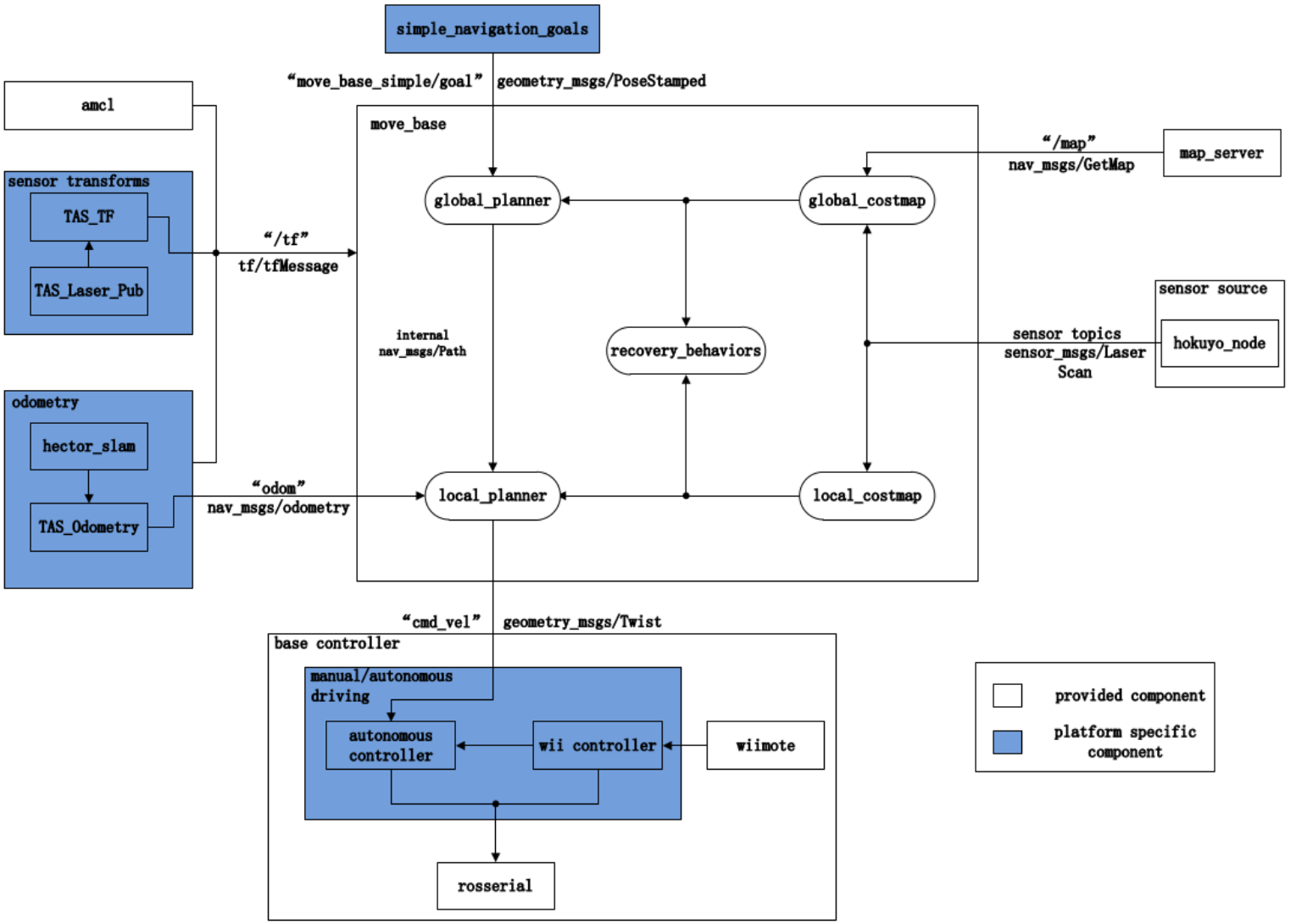
Topics

- A node sends out a message by publishing it to a given topic
- The topic type is dependent on the message type published on it
- When a node requires a certain type of data it should subscribe to a the correct topic
- There can be many-to-many and one-to-many subscription/publishers
- Publishers do not care if there are subscribers and subscribers don’t really care how publishers work… as long as they get their message at the end of the day
- This eliminates the need for an order of execution (asynchronous)

Services

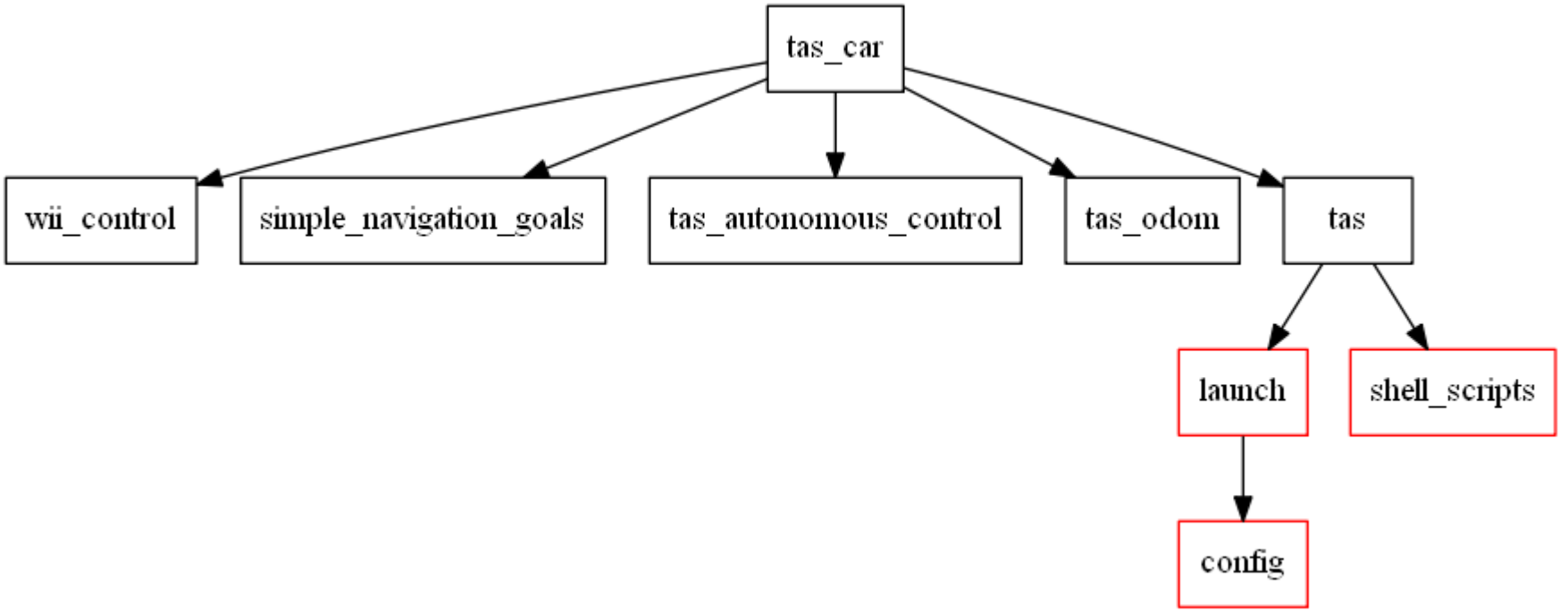
- For more synchronous (ordered) communication, we have the option of using services.
- It has the request/reply functionality
- A node will send a request message and wait for the reply
- The other node offers this service
- There are a pair of message structures, one for the request, one for the reply
- This is similar to a remote procedure call
- Topics, Services, Actions
  - Actions are asynchronous with feedback monitoring

TAS Car System Architecture

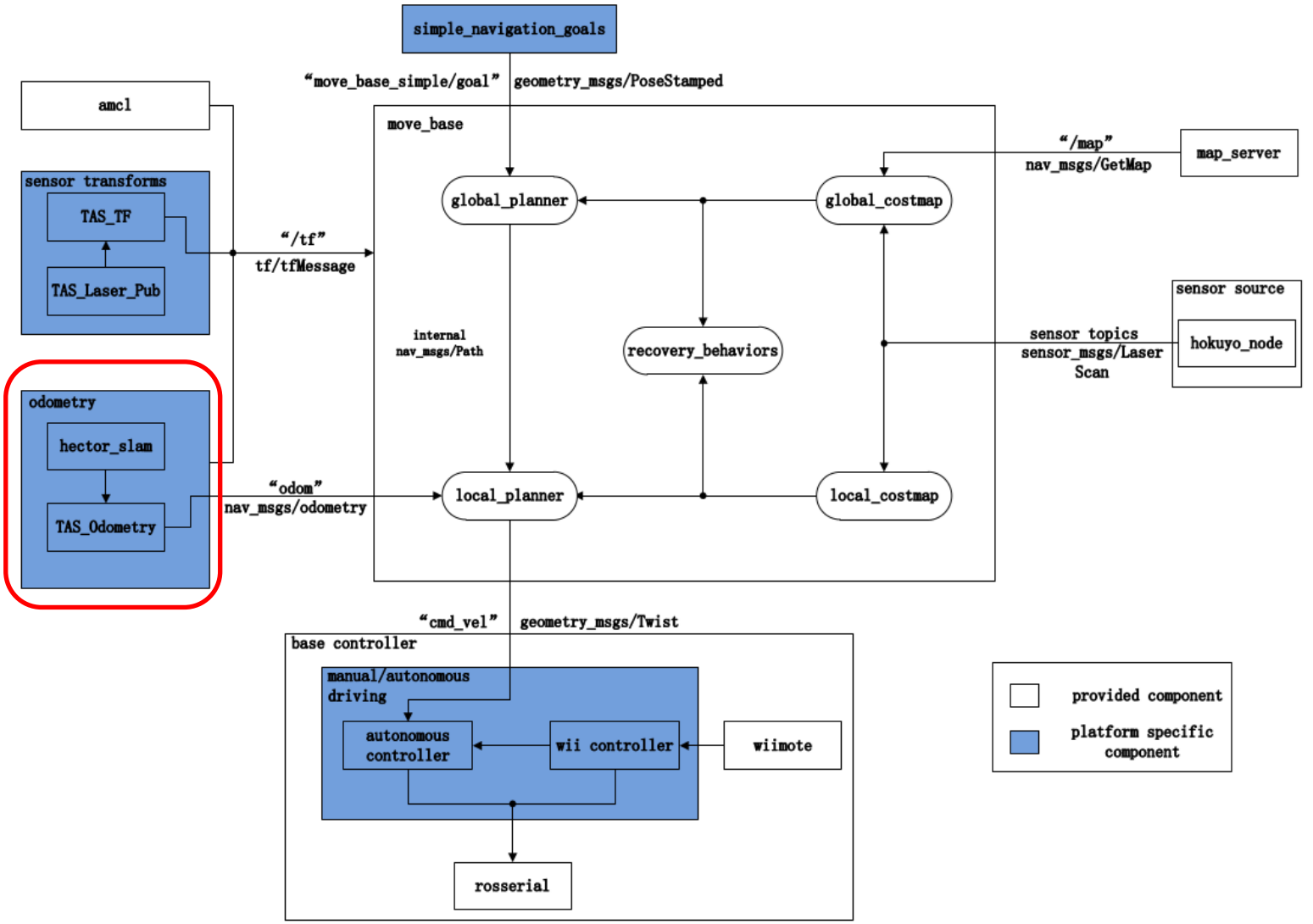


Tas_car Package

- Git clone https://github.com/LSR-TAS/tas_car



TAS Car System Architecture

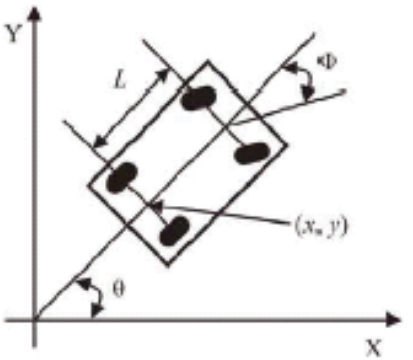


Odometry Source

- Goal
  - Transformation from “fixed" /odom frame to movable /base link coordinate system
  - Wait.. What are those frames?!

- Possible approach
  - Use available wheel odometry (rotation sensor), orientation (inertial measurement unit) and state space model of the car-like robot (bicycle model)

$$
\begin{aligned}
x_{k+1} &= x_k + dT \cdot v_k \cdot \cos(\theta_k) \\
y_{k+1} &= y_k + dT \cdot v_k \cdot \sin(\theta_k) \\
\theta_{k+1} &= \theta_k + dT \cdot v_k \cdot \tan(\phi_k)/L
\end{aligned}
$$



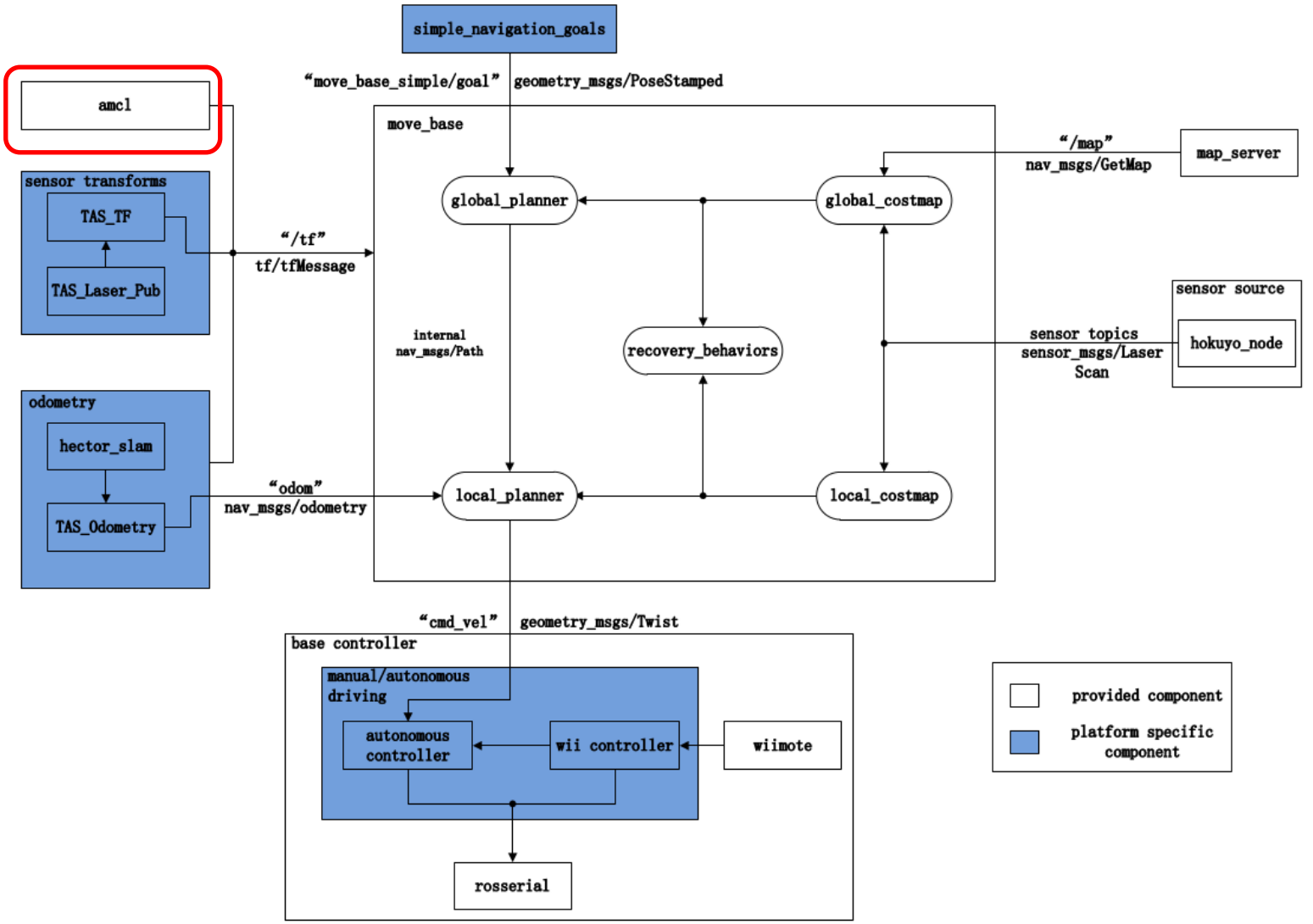
Odometry Source

- Problems with initial approach:
  - Highly unreliable due to inaccurate wheel odometry

- What we have now:
  - Use Laser SLAM (hector_mapping) to fake odometry information based on laser scans … but as it depends on laser scans it has low update rate ..

- We installed an IMU so now we have other possible approaches:
  - Extended Kalman Filter to combine model-based odometry with estimated laser scan matcher-position
  - The above solution is really the first thing you should be working on as you need angular/ linear velocity, angular/ linear positions estimates.

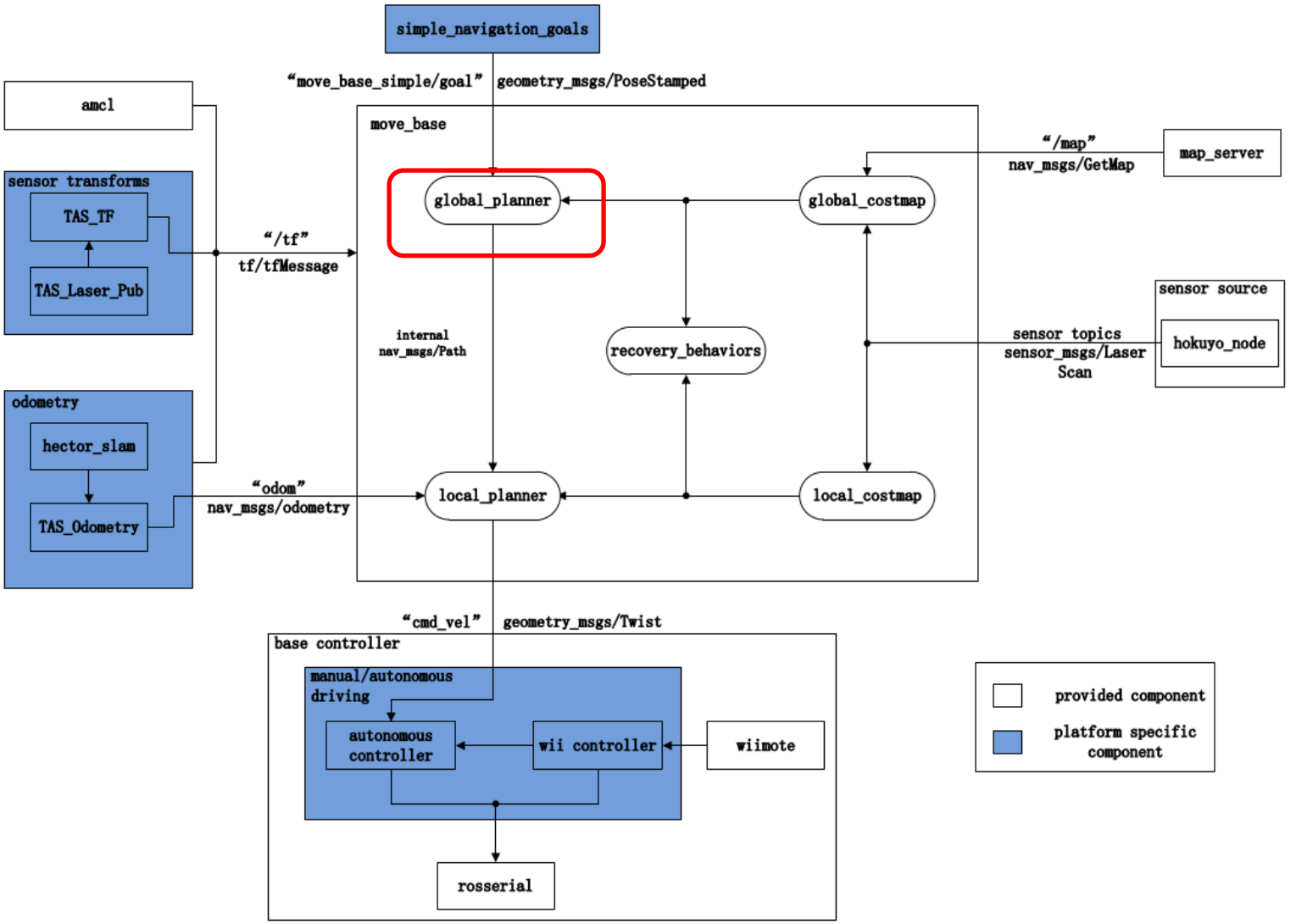
TAS Car System Architecture



Adaptive Monte Carlo Localization

- Goal
  - Localize robot in a given 2D map based on laser scans and transform messages
- Advantages
  - Adaptive size of sample sets
  - Nonparametric particle filter
- Difficulties and Limitations
  - Limited computational power
  - Map issues
  - Parameter setting

TAS Car System Architecture



Global Planner (nav_core)

- Goal
  - Generate a feasible global path from current to goal position
- ROS global planner that use the nav_core::BaseGlobal Planner interface:
  - navfn - A grid-based global planner that uses a navigation function to compute a path for a robot. Navigation function is computed with Dijkstra's algorithm, but also can use A* heuristic.
  - carrot_planner - A simple global planner that takes a user-specified goal point and attempts to move the robot as close to it as possible, even when that goal point is in an obstacle.
- Limitations:
  - Non of these necessarily take the kinematics/ dynamics of the car into account

Alternative: SBPL-Lattice Global Planner

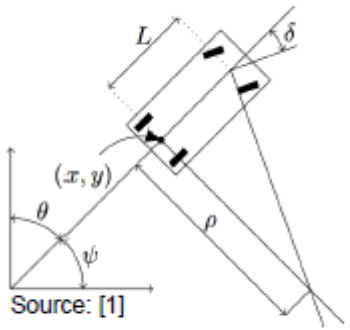
- Goal
  - Generate a feasible global path from current to goal position using short, kinematically feasible motions which form the basis of movements that can be performed by the robot platform known as primitives. Search based planning librarby …
- sbpl_lattice_planner - (x,y,yaw) planning for robot navigation and has the advantage of handeling nonholonomic constraints

- Is a good option if one takes the current car kinematics into account and define a set of possible primitives that the local planner can surly achieve/control

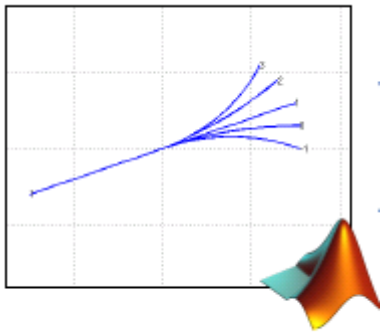
- Check this link out for a tutorial: http://sbpl.net/Tutorials

Alternative: SBPL-Lattice Global Planner

- However, you have to define the motion primitives



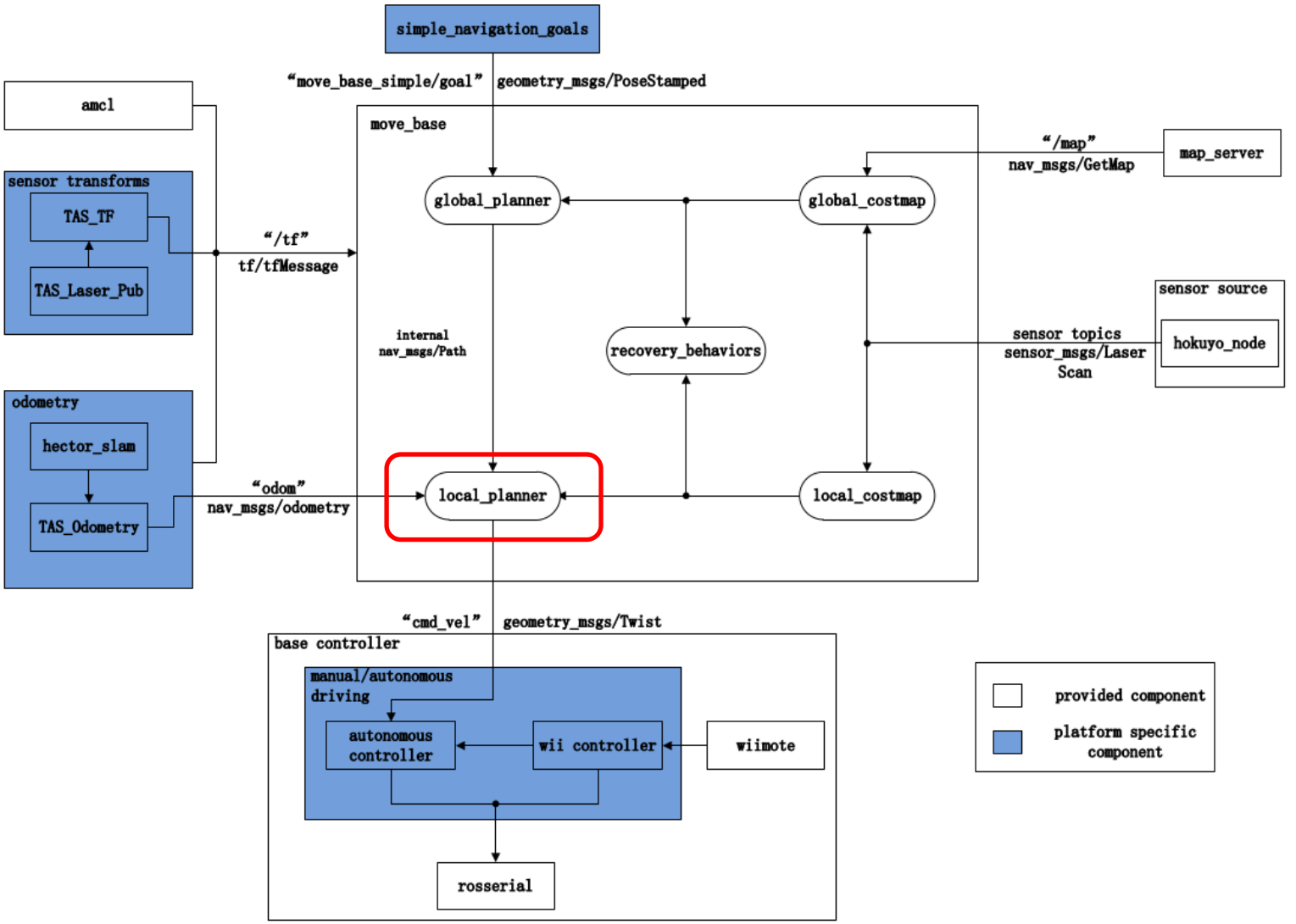
Vehicle Model



Motion Primitives

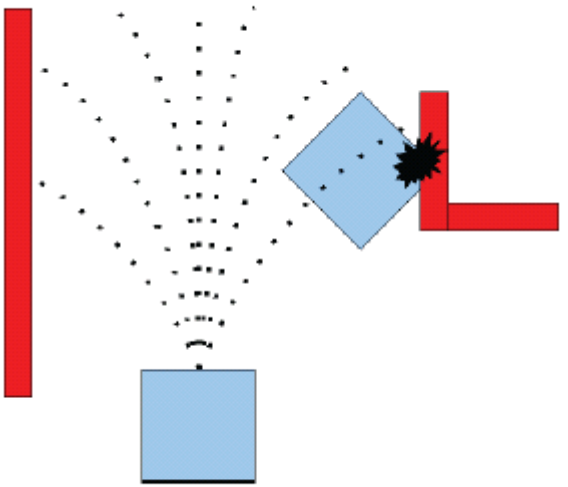
Global Path

TAS Car System Architecture



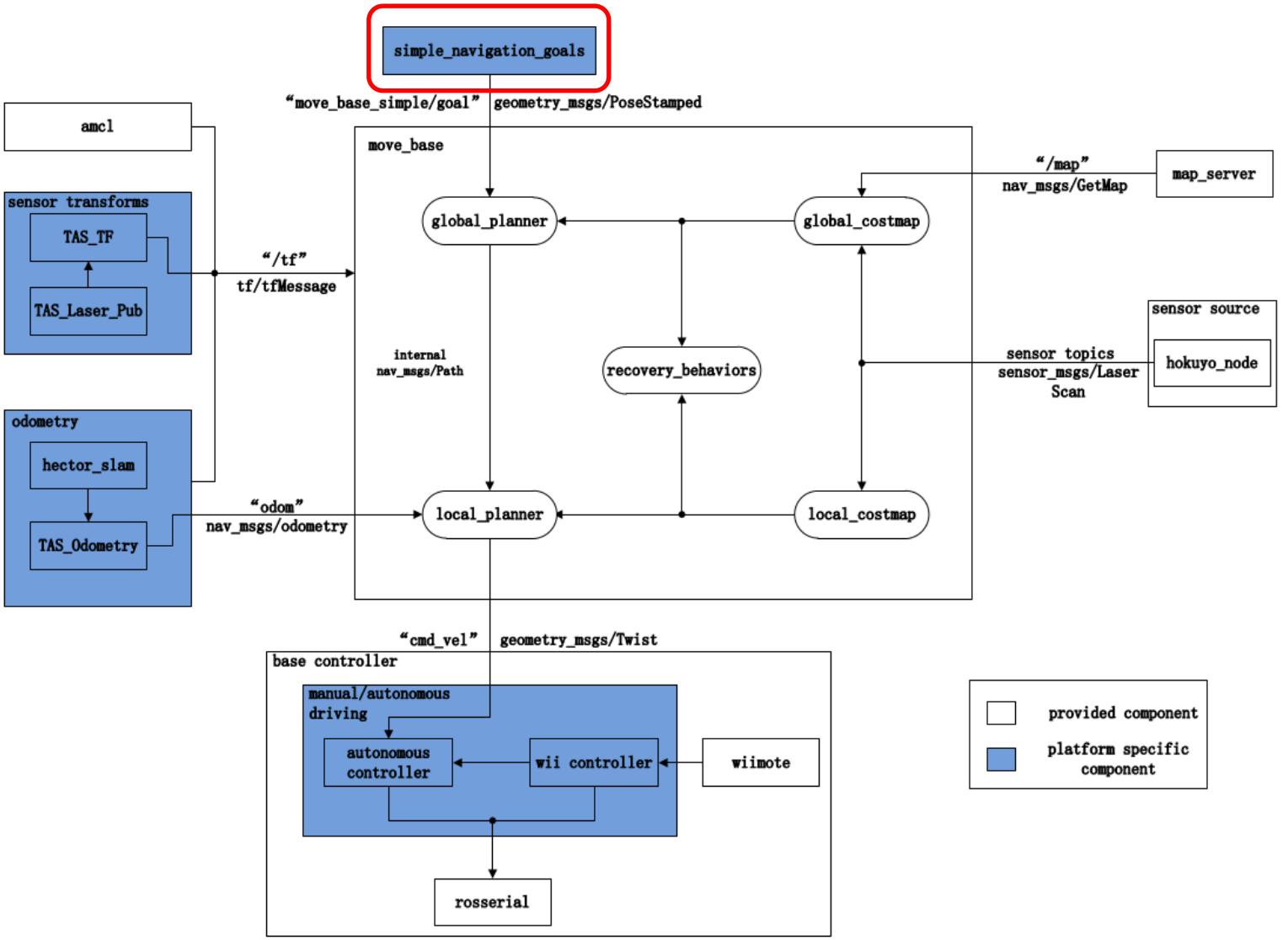
Local Planner

- Goal
  - Local path planning
  - Uses trajectory roll-out and local cost map
  - Tries to stay close on the global path



- Advantages
  - Evaluates (score) trajectories resulting from forward simulation
  - Uses the highest-scoring trajectory to send the associated velocity (dx, dy, dtheta) to the robot
  - Checks for collisions using the footprint of the robot

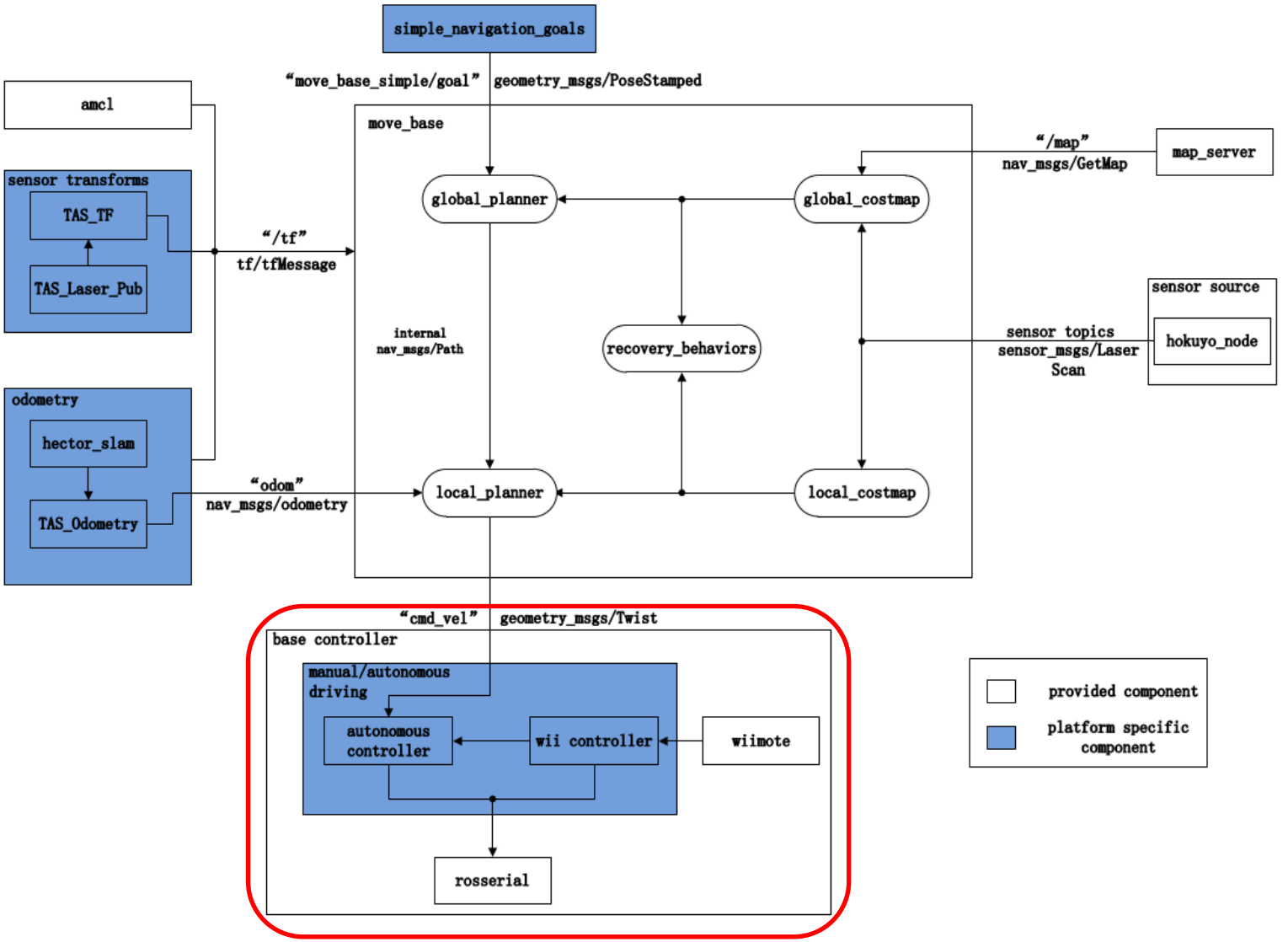
TAS Car System Architecture



Navigation goals

- Goal
  - This package is for setting the navigation goals for the RC car during the autonomous driving.
  - After the current navigation goal position is achieved, the robot will go to the next goal position.

TAS Car System Architecture



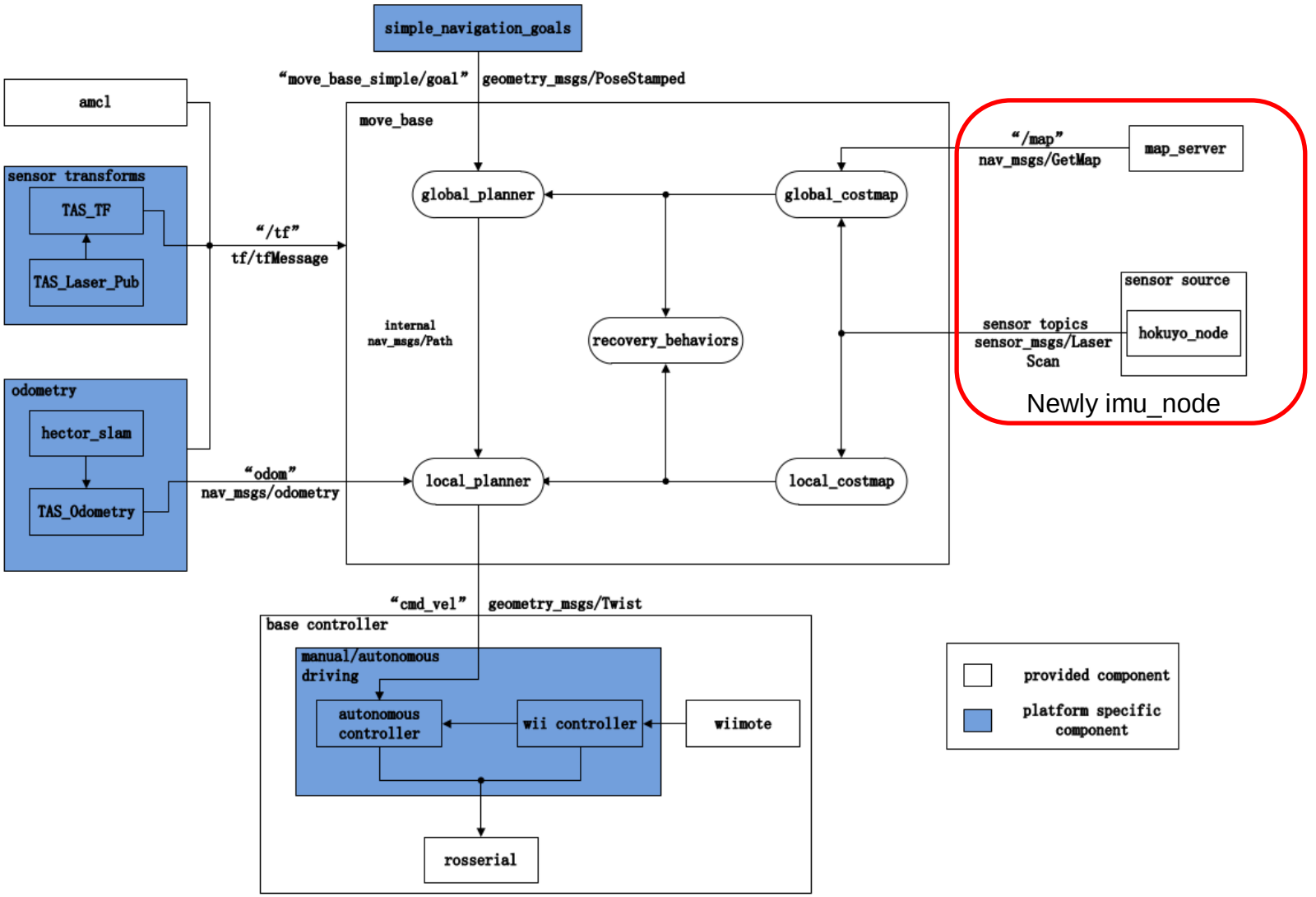
Base Controller

- Actuate wheels and steering as given in /cmd vel from move base
- This package serves as a high level controller for the car.
- In autonomous mode, it receives the cmd messages from move_base, and then converts the cmd messages to servo messages based on car kinematic model and publishes them to topic /servo.
- Currently, mapping for steering angel is finished and the velocity of the car is not continuous but preset to specific values.
- Later, an improvement could be the implementation with IMU for velocity feedback control.

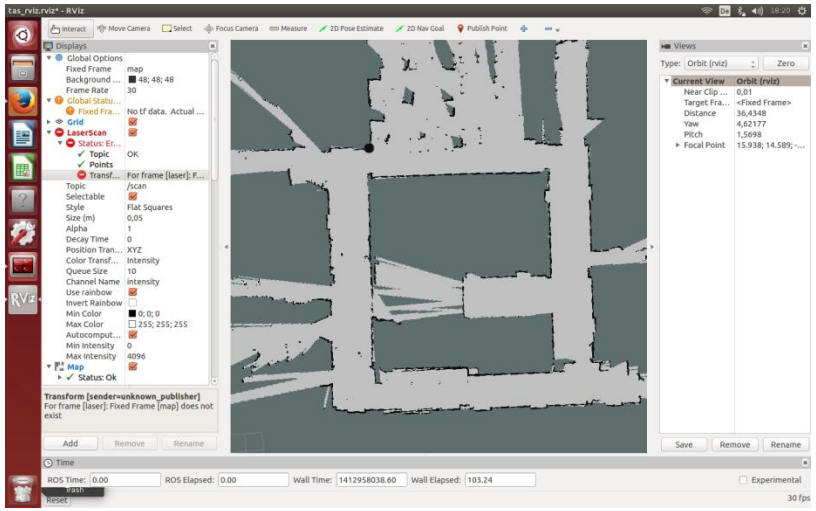
Wii Controller

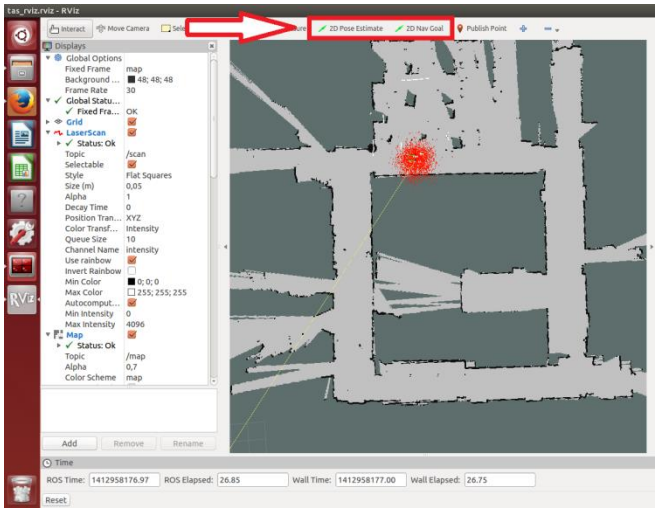
- This package is for manual control of the car with wiimote!
- It subscribes to the wiimote states and maps the analog button of Nunchuk into servo messages.
- This package also determines the control mode of RC car and sends the control mode state using flags to the autonomous control node.
- By pressing button C of Nunchuk, the control mode will be switched to autonomous driving. After releasing button C, the control mode will be automatically reset to manual driving.
- Button Z of Nunchuk is for resetting servo messages.

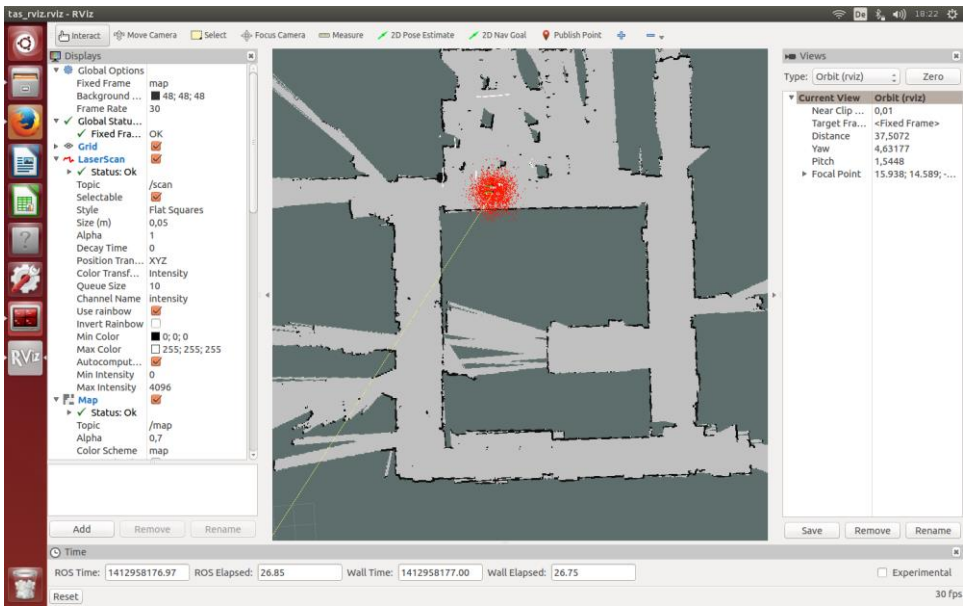
TAS Car System Architecture



Let’s Look How This is implemented on the Car







How to get started…

- Send an email to khoi@lsr.ei.tum.de & laith@lsr.ei.tum.de with your group member names and emails + which 2nd task you will do…
- ROS tutorials first then Navigation Stack tutorials …
- Download and understand the git package tas_car
- Visit Khoi for a discussion about your understanding of ROS/Navigation and your ideas for algorithms …
  - Git clone https://github.com/LSR-TAS/tas_car.git
- Get your password for the cars and the car manual
- Setup a github account for your group -xx stands your group # 01-14:
  - Username: tas-group-xx
  - Repository name: tas_car_xx
- Start adjusting your copy of tas_car
- Book a date for
  - https://docs.google.com/spreadsheets/d/1Yoc4rOhf1kwlUR7E6a0t1KsIQE0yoUaw_NH8hskekHc/edit?usp=sharing
  - Each group is limited to 12 1-hour slots a week, if no reservations .. Continue☺

What will you be graded on…

- A scientific contribution (personal) … this means you have to read on the topic you are working on
- A scientific contribution
- CONTRIBUTION
- Completion of the lap around N5/3rd floor and how you rank in terms of speed …
- Completion of the additional task
- Time management
- Your (personal)git commits and how well your code is maintained/commented
- New functionality beyond state of the art

Contact and Discussion

- Please contact Khoi khoi@lsr.ei.tum.de for organizational/ and issues with the practical course
- You will be given a manual that answers almost all your questions about the car and tasks ..
- Questions?