Documentation roburoc4_aseta

This document contains the documentation of the implemented of the RobuROC4 setup at AAU.

Document created: 25. October 2012
Last edited: 17. November 2012

Dependencies:
C-libraries:

Armadillo:	Matrix manipulations (remember to install lapack beforehand!)
Geographiclib:	Conversion from Lat/Long to UTM
Gnuplot:	Plotting figures (calibration of magnetometer)

ROS-packages:

joy:	Interface to joysticks (including ps3_controller)

Other:

qtsixa:	Program (“Driver”) used to interface the ps3 controller such that the joy ROS package can find it

In this documentation file each packet will be described in terms of nodes included in the package and what the in and outputs of the nodes are (topics). When the topics are described the message type used on the topic is shown in paranthesis.

The most of this package is implemented during two semester projects at Aalborg Univercity by project groups 11gr731 and 12gr832. See project reports in the documentation folder!

Quick start guide

To get the roburoc4 up and running the only thing you should do is to make sure that the root of the roburoc4 stack is in the ROS_PACKAGE_PATH. And the code is compiled. To compile all code for the aseta_roburoc4 system run the following command:

rosmake aseta_bringup

If the system compiles i can be started in different modes:

1. Start Roburoc4 driver, sensors, state_estimator, and controllers:
   roslaunch aseta_bringup roburoc4.launch
2. Start Roburoc4 driver, sensors, state_estimator, controllers, and onboard menu:
   roslaunch aseta_bringup roburoc4_w_menu.launch

For other automated ways to start the system look into the aseta_brinup package described below. For each package one (or several) launch files are available which could be used in only a part of the system should be started. i.e. If you want to add teleoperation functionality toy could start one of the teleoperation nodes by using the launch files in the kdh_teleop package. For more info look at the separate packages.

aseta_bringup

This package is used to compile and launch the whole system. The package does not contain any functionality other than containing a compilation of launch files to start the roburoc4_aseta system in different setups. The package depends on all the other packages. The main reason for this is that making the aseta_bringup package using rosmake makes all packages in the roburoc4_aseta stack.

launchfile(s): **roburoc4.launch** Launch the roburoc4 system, including roburoc4 driver, sensors, state estimation, and controllers.
roburoc4_w_menu.launch Launch the roburoc4 system, including roburoc4 driver, sensors, state estimation, controllers, and the onboard menu.

aseta_sensors

This package contains the interface to the GPS, Magnetometer, and Gyro. For each sensor a node is implemented. The communication to all the sensors are serial RS323 connections with different protocols for each sensors.

nodes: **magnetometer_node**, **gps_node**, and **gyro_node**

launchfile(s): **aseta_sensors.launch** (Launches all three nodes)

topics (Sub): **Magnetometer/calibrate** *(std_msgs/Empty):* Publish a message to this topic to initiate calibration of the magnetometer. Magnetometer must be rotated during calibration (follow instructions shown in the terminal where the nodes are started).

topics (Pub): **GPS/position** *(sensor_msgs/NavSatFix):* GPS position lat/long in decimal degrees, number of satellites, and RTK status.
GPS/orientation *(geometry_msgs/Pose2D):* Heading measured by the GPS. Heading is in radians (between 0 and 2*Pi) with positive direction counter clockwise and 0 to east! (Made as this to fit the UTM coordinate system).
GPS/velocity *(geometry_msgs/Twist):* The linear velocity (m/s) measured by the GPS.

Magnetometer/orientation *(geometry_msgs/Pose2D):* Heading measured by the magnetometer. Heading is in radians (between 0 and 2*Pi) with positive direction counter clockwise and 0 to east! (Made as this to fit the UTM coordinate system).

Gyro/velocity *(geometry_msgs/Twist):* Angular velocity (rad/s) measured by the Gyro.

roburoc4_controllers

This package contains the two different controllers, a helmsman controller and a trajectory controller. For more information about the principle about those two controllers refer to the following project reports in the documentation folder:

- helmsman: project_reports/11gr731_line_tracking_worksheets.pdf
- trajectory: project_reports/12gr832_robotic_road_striper.pdf

In addition to the two controllers functionality for logging references (line segment waypoints for the helmsman controller and trajectories (x(t),y(t),theta(t),omega(t),v(t)) for the trajectory controller)

nodes: **roburoc4_helmsmann_controller, roburoc4_trajectory_controller,** and **roburoc4_trajectory_logger**

launchfile(s): **roburoc4_helmsmann.launch**, **roburoc4_trajectory.launch**, and **roburoc4_trajectory_logger.launch**

topics (Sub): **roburoc4_state_estimator/states** (*roburoc4_state_estimator/States*): The current states of the vehicle. Used in the controllers to generate control signals. Used in the logger to capture the references for the controllers.
roburoc4_controllers/controlMode (*std_msgs/UInt8*): Used by both controllers to see if it should be active or not. One control mode is defined for each controller. If the control mode is 0 all controllers are disabled!
roburoc4_controllers/lineSegment (*roburoc4_controllers/LineSegment)*: Used by the helmsman controller to get the reference to follow.
roburoc4_controllers/trajectory (*roburoc4_controllers/Trajectory*): Used by the trajectory controller to get the reference to follow.
roburoc4_controllers/logger_cmd (*std_msgs/UInt8*): Used to command the logger to capture trajectory or capture start and end points for the line segments to the helmsman controller.

topics (Pub): **cmd_vel** (*geometry_msgs*): Used to command the RobuROC4 to drive with a specific velocity.
roburoc4_controllers/lineSegment (*roburoc4_controllers/LineSegment)* and **roburoc4_controllers/trajectory** (*roburoc4_controllers/Trajectory*): Published by the logger to give the logged values as reference.

parameters: **/roburoc4_controllers/recorded_trajectories** Used to specify the folder in which saved trajectories must be saved to and read from.

Notes on the logger functionality: To make logged trajectories available after a reboot of the system the logged trajectories are saved in files in the folder specified in the mentioned paramerer (if any folder is defined). If a folder is defined the node tests if the following four folders exists:

- helmsman
- helmsman-use_this
- trajectory
- trajectory-use_this

If a folder is missing they are created! When a recorded trajectory it is saved in either the helmsman or the trajectory folder depending on what type reference is logged. The files are

saved in the following format: YYYY_MM_DD_HH_II.txt. When the trajectory logger is started it browses through the helmsman and trajectory folders and loads the most recent modified file in those folders. The two use-this folders is used to overrule the two other folders. If a file is put into those folders that file will be used! If multiple files is saved in the use-this folders the no rule is defined for which file will be used. Therefore: **put only one file in the use-this folder!** and do never add the recorded-trajectories folder to things such as svn where hidden files are saved in the foldes since this will ruin the structure!

For the helmsman logging end point and start points are logged separately and could be logged in arbitrary order. The points are saved manually by calling the save command, and published manually be calling the command for that!

For the Trajectory logging, the logging is started by a command, and ended by another command. When the logging is ended the trajectories are cropped such that the ends where the velocity is very close or equal to zero are removed. After cropping the trajectories are automaticly saved. The trajectory is published by using the command for that!

roburoc4_menu

The menu is a terminal based gui used start and stop the controllers and visualize what is going on the robot (showing sensor inputs and estimated states). The menu is intended to be used on the onboard computer but could equally well be opened on a remote computer, using the ros core on the onboard computer!

nodes: **roburoc4_menu**

launchfile(s): **roburoc4_menu.launch**

topics (Sub): ***all the sensor topics, and the topic containing the estimated states.***
roburoc4_printString (*std_msgs/String*): Contains user relevant information published by other nodes. Is printed in the status field in the menu.

topics (Pub) **roburoc4_controllers/controlMode** *(std_msgs/UInt8):* Set the control mode.
roburoc4_controllers/logger_cmd *(std_msgs/UInt8):* Send commands to the logger node (i.e start/stop logging, and publish trajectories to controller)
Magnetometer/calibrate *(std_msgs/Empty):* Command the magnetometer to initiate calibration.
roburoc4_state_estimator/reset_filter *(std_msgs/Empty):* Command state estimator to reset the filter

roburoc4_state_estimator

The state estimator is used to fusion sensor measurements from all sensors using a model of the robot. To carry out the estimation of the states an extended kalman filter is used. For more information about the extended kalman filter refer to:
project_reports/12gr832_robotic_road_striper.pdf in the documentation folder.

nodes: **roburoc4_state_estimator**

launchfile(s): **roburoc4_state_estimator.launch**

topics (Sub): **cmd_vel** *(geometry_msgs/Twist): Used as model input in the filter!*
roburoc4_driver/cur_vel *(geometry_msgs/Twist):*
GPS/position *(sensor_msgs/NavSatFix):*
GPS/orientation *(geometry_msgs/Pose2D):*
GPS/velocity *(geometry_msgs/Twist):*
Magnetometer/orientation *(geometry_msgs/Pose2D):*
Gyro/velocity *(geometry_msgs/Twist):* All used as measurement inputs!

topics (Pub) **roburoc4_state_estimator/states** (*roburoc4_state_estimator/States*): The current states of the vehicle estimated by the state estimator.

roburoc4_driver

The roburoc4_driver packet contains the interface to the RobuROC4 four wheeled skid steered vehicle. The roburoc4_driver only contains one node which merely implements the pure protocol, the communication protocol used to communicate with the RobuROC4 via a UDP interface.

nodes: **roburoc4_driver**

launchfile(s): **roburoc4_driver.launch**

topics (Sub): **cmd_vel** *(geometry_msgs/Twist):* Use to command the RobuROC4 to drive with a specific velocity.
roburoc4_driver/req_properties *(std_msgs/Int32):* Use to get the properties of the service with the service id given as argument. The response is shown as text in the terminal where the roburoc4_driver node is started.
roburoc4_driver/set_digital_output *(std_msgs/UInt8):* Use to set the digital outputs 0 to 7 corresponding to the value of the 0'th to 7'th bit in the unsigned char given as argument.
roburoc4_driver/set_cur_pos *(geometry_msgs/Pose2D):* Update the internal position estimate of the RobuROC4.

topics (Pub) **roburoc4_driver/cur_vel** *(geometry_msgs/Twist):* The current velocity of the vehicle
roburoc4_driver/cur_pos *(geometry_msgs/Pose2D):* The current estimate of the position made by the RobuROC4
roburoc4_driver/battery_info *(roburoc4_driver/battery)*: Remaining power and charging state of the battery
roburoc4_driver/io *(roburoc4_driver/io):* State of all analog and digital inputs.
roburoc4_driver/telemeter *(roburoc4_driver/telemeter):* Distance to an object from one of the two telemeters in the RobuROC4.

kdh_teleop

Packet containing two teleoperation nodes. One for using joystick and one for using a ps3 controller. It should be noted that qtsixa must be installed and working before the teleop_ps3_controller node will work!

joystick: For controlling by using a standard joystick (by Karl Damkjær Hansen)

ps3 controller: For the ps3 controller the two analog joysticks are used to control it and R2 is used as deadman button (if not pressed the node spits out zero velocity!). The node implements two different modes:

1. One stick control. The vehicle in controlled by moving the right joystick forward/ backward movement control the linear velocity output while sideways movements control the angular velocity.
2. Two stick control. In this mode only forward/ backward movements are used from the two sticks. Left and right stick controls the velocity of left and right wheelpair respectively.

nodes: **teleop_joy, teleop_ps3_controller**

launchfile(s): **teleop_joy.launch** (Launches teleop_joy)
teleop_ps3_controller.launch (Launches teleop_ps3_controller)

topics (Sub): **joy** *(sensor_msgs/joy):* Get inputs from the connected joystick / ps3_controller. (Provided by the ROS joy package, launched by the launchfiles)

topics (Pub): **cmd_vel** *(geometry_msgs/Twist):* Use to command the RobuROC4 to drive with a specific velocity.

parameters: **angularAxis** and **linearAxis** Used to specify the desired axis used to control the angular and linear velocity when using teleop_joy.
angularScale and **linearScale** Used to specify the maximum angular and linear velocity when using the teleop_joy.
deadManButton Used to specify a deadman button for the teleop_joy.

/teleop_ps3_controller/angularScale Used to specify the maximum linear velocity when using the ps3 teleop.
/teleop_ps3_controller/linearScale Used to specify the maximum linear velocity when using the ps3 teleop.
/teleop_ps3_controller/deadManButton Used to specify which button should be used as deadman button (default: 13 = R2) when using the ps3 teleop.
/teleop_ps3_controller/controllerMode Used to specify desired mode (1 or 2) when using the ps3 teleop.