Control Strategy

The System uses a two staged control system (Nick's figure from the group meeting).

What is it?

The control system is the heart of the autonomous navigation, it is responsible for keeping the ship on its predefined course. The inputs of the whole control system are the sensor measurement data, and the output is the motor speed PWM percentages, passed to the LLI.

What are we using?

We are using a three-staged control system that divides the main controller to two sub-controllers and a conversion matrix, based on their functionality.

In the First Stage the required heading of the ship is calculated that keeps the course closest to the planned waypoint.

In order to efficiently and accurately navigate along the path, a set of Sub-Waypoints is calculated for each route between two Waypoints. The main control strategy of the first path is to navigate through all of these SWPs in a predefined order, one by one. (Navigation Figure) The heading of the ship is defined in NED coordinate system. The required heading is determined by the law of Cosines, based on the Position of the Ship and the Position of the next Sub-Waypoint (Law of Cosines Figure). Problems rise and corrections are necessary, if the heading of the ship θ is θ < -π or π < θ. The heading of the ship is calculated based on the Gyro sensor and the heading can have any value in the form of: θ = [-π;π] +- 2\*k\*π. Before invoking the control procedure, all of the heading angles must be transformed into the [-π;π] interval.

This procedure causes an error though, shown in (Heading Figure). The required heading or the heading of the ship must be transformed into a different representation, where θr-θ < π. To keep a consistent heading representation, first the deviation angle δ = θr-θ is calculated, than transformed to the [-π;π] interval and finally, with δ we can transform θr to θr(θ) = θr – δ.

If the conditions above are met, the θ and θr(θ) will always yield values that result in correct controller output.

The overall navigation can be improved, if the Target SWPs are considered “reached” in a certain distance. Optimally this distance (Validity distance) should be somewhat longer than the minimum turning radius of the ship (Rmin). If it’s shorter, the ship might not be able to reach the SWP, if it’s way longer, the ship will divert from its course in turns. Generally if the Validity distance is approaching Rmin the ship will stay close to the Waypoints, and will navigate through a geometrically broken line, but if the ship speed is too high, the overall path-accuracy might be low. At a higher speed a longer Validity distance is recommended, because it will increase the path-accuracy in the overall navigation. (Validity Figure)

Why are we using it?

The main objective of the controller is to provide a fairly robust system that is unable to enter an unstable or uncontrollable state. In order to ensure this, we have analyzed all of the possible ship states, and crated a simple but efficient method to control the ship. This can be done by guaranteeing that the input of the second stage is always appropriate, under any circumstances. We secured this condition with the proper pair of θ and θr, resulting in the SWP Validity checking and the conditioning of θ and θr.

The Second Stage ensures that the ship will stay true to its required heading, therefore staying on its local path.

\*Description of State-Space Controller\*

\*Description of Transfer matrix\*

Path Planning

Course keeping control

What is it?

The Course-keeping control is responsible for keeping the ship on-course during Autonomous navigation.

What are we using?

The general idea is to have a state-space control algorithm in infinite loop as the main task. The control parameterization is based on the ship model, measurements and identification.

The program controls the ship along the specified path. If there is no next sub-waypoint or local path specified, the HLI calls the path planning methods for the next waypoint, then the procedure starts again from the beginning. If there are no more waypoints, the ship returns to the first waypoint, or to the starting coordinates specified in a subfunction of the Ship class.

Why are we using it?

Using Sub-Waypoints instead of a full path line makes the navigation much easier. In open water there is significant sideways-motion caused by wind, ocean currents and waves, so staying perfectly true to a predefined path line is extremely difficult.

Instead, by using a series of Sub-Waypoints, the navigation of the ship resembles to a series of Buoys guiding trough a causeway. The Ship approaches each Sub-Waypoint in a predefined order and turns to the next, when the “buoy” is close enough.

The characteristics of the navigation and course can be varied only by changing the definition of the Sub-Waypoints, setting optimal parameters for different locations and settings.

Waypoint Planning:

What is it?

The waypoint-planning system provides the basic routing for the autonomous navigation system.

The purpose of the waypoint-planning is to set key coordinates that the ship must approach, either for strictly defined reasons or to keep the ship away from un-sailable or dangerous areas (like the coast or an island)

What are we using?

There are two possibilities to set a collection of Waypoints: The operator can either manually set them, based on the GPS coordinates of the Waypoints, or by inputting a coastline data series is a specified data type. The coast input format is a series of perpendicular distances measured from a line parallel to the coast. The coast input can be generated based on map data (Openstreetmap).

The automatic waypoint planner divides the coast to smaller parts, based on the required oceanography definition. For each segment a minimum approach distance is calculated, and the waypoint planner defines a set of waypoints along the path in a snake way, or in any other predefined setting. The figure shown a simulation of an oceanography task using automatic waypoint planning. All values in meters. (Auto\_WP\_Planning figure)

After the ship visited all of the waypoints, it returns to the first waypoint, or to a specified return position.

Why are we using it?

The need for a basic Waypoint-planner is essential, but optimizing it was not a high priority task. To test the prototype, a series of manually set waypoints are adequate.

Local planner:

What is it?

The Local planner is responsible for planning a segment of the path, in order to supply the controller with a series of Sub-Waypoints. The Local path should result in a set of points, which are lined up smoothly enough for the ship to sail through them with the reference speed.

What are we using?

The path can be divided to a straight line and a turning sub-path. The combination of a straight path and a turn is a Path Segment. The local planner calculates a Path Segment from the current position to the end of the turn at the next Turning Waypoint (Footnote: If three waypoints are in a line, the angle at the second is π, therefore it’s not a turning waypoint), based on the Waypoint after the Turning Waypoint as well.

Calculating the smooth set of points for the turn:

The initial idea was to use a pre-generated Euler-spiral or simple arc line, the program determines the required arc length and applies linear transformations to resize and rotate the path coordinates, thus creating a smooth path.

The initial idea was dropped after the following considerations:

* Storing the full path coordinates is memory-consuming. Also, the path has either low resolution or the linear transformation would be CPU-heavy
* The path would be smooth but would not have optimal parameters

The new idea is originated from: Komlósi István: Mobilis robotok autonóm navigációja mozgó akadályok elkerülésével (English version: István Komlósi and Bálint Kiss: Motion planning for multiple mobile robots using time-scaling)

The idea is to determine the maximum possible path curvature that the robot can handle. This is based on the Sigma and Kappa values of the Ship, where sigma is a function of the maximum speed of Torque change, and kappa is the maximum curvature of the path at a given speed. The local path is generated in a specific way that the robot will always turn with the maximum possible curvature at the current speed, thus staying the closest to the waypoint without losing speed. There is a threshold turn-angle, which determines if the robot requires only two identical Euler-spiral paths to turn or an arc path that has the maximum curvature, with two Euler spiral paths leading in or out.

In order to create the path the algorithm calculates 5 or 3 (depending on the threshold) key points and fits a Hermite-poly onto them (Figure). From this point on the local path in the given range is determined by these points only, thus saving memory and CPU time, while calculating a better path.

The calculation of the key-points is detailed in (Figure).

The Hermite-polinom is used to populate the path with a number of points, depending on the predefined conditions. The resulting set of points is transformed to its correct place by a transformation matrix in the Local Frame, around the Turning Waypoint. (Figure)

Why are we using it?

The considerations behind this path planning method were based on the following conditions:

* The ship must be able to output the next waypoint quickly, therefore calculating the whole path line in a single batch was to be avoided
* The Waypoints of the ship are subject to possible changes. Re-calculating the whole path every time a Waypoint is changed is very consuming.
* The Ship is subject to unpredictable outside forces. Every path-segment is to be computed to be optimal, based on the actual, not the ideal prepared position
* The local planning should be as efficient as possible. Planning every Sub-Waypoint individually is a lot less effective than planning them in a batch.