Handling lateral tracking error #2859

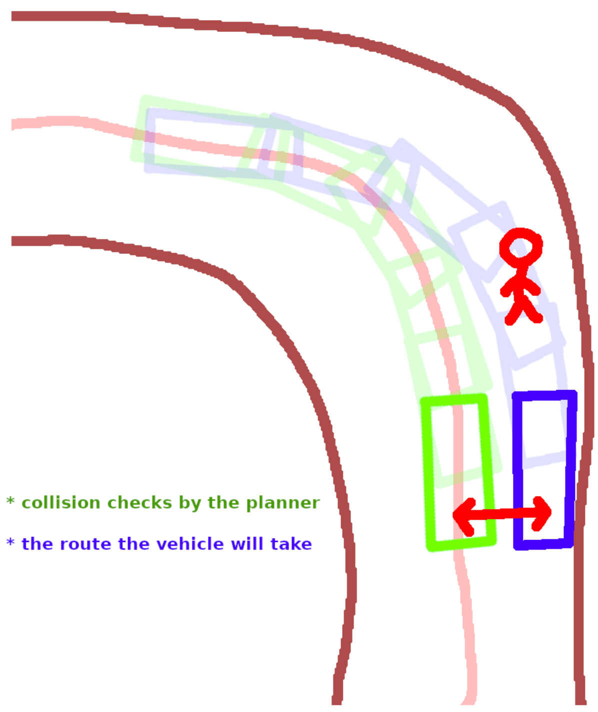
xmfcx started this conversation in Design

xmfcx on Sep 13, 2022 Maintainer

Scenario

- The vehicle is making a turn.
- Planner creates a path.
- Controller isn't able to follow it.
- Vehicle starts diverging from the center.
- Collision checks are still being done in the planner, assuming the vehicle is on the path.
- Vehicle ignores obstacles in front of it.
- It possibly crashes.

Simple illustration


* collision checks by the planner
* the route the vehicle will take

Countermeasures taken in Autoware Universe

- Use lane departure checker to send an emergency stop command if vehicle has exceed lane departure threshold.
- MPC has an option to check for obstacles in its planned horizon.

Category

Design

Labels

component:planning
component:control

6 participants

Suggested solutions

- Predict trajectory of the vehicle, based on vehicle's velocity and steering angle to check for collisions. (by Brian from ITRI)
- Tune the controller so this never happens.

@TakaHoribe @mitsudome-r @brian841102 @mehmetdogru any other ideas?

↑ 3

6 comments · 14 replies

Oldest | Newest | Top

mehmetdogru on Sep 13, 2022 Maintainer

@xmfcx

Well currently we use pure pursuit as lateral controller. Since there is no predicted path calculated whatsoever in pure pursuit I think we can implement the idea of predicting a path with current velocity and steering angle for x seconds. It makes sense to me. It is simple and can be implemented fast.

Then we can check if it collides with an obstacle and if it will drive outside of the drivable area.

↑ 1 👍 2 0 replies

xmfcx on Sep 13, 2022 Maintainer Author

I have a suggestion too on this case.

- Planner calculates both of these routes.
- One for reference path on the center line
- One for extra collision checking, starting from base_link

Advantages

- This would be more reliable than blind prediction from the vehicle's velocity and steering angle.
- This wouldn't prevent planner providing a stable reference for controller to follow.
- Obstacle avoidance would be strictly handled in the planning stack, not leaking down to control stack.

↑ 1 0 replies

yukkysaito on Sep 13, 2022 Maintainer

The current design makes the following assumptions as far as I know

- The ideal goal of control module should follow on trajectory without error. Also, the motion planner module should generate a trajectory that can be followed.
- If a lateral error occurs in control module, the possible error should be set in the motion velocity module.

However, I do not think it is good to have such an implicit understanding between planning and control.
Therefore, I think that the acceptable control error should be included in the type of trajectory. The control module should be able to run within an acceptable control error. If it cannot run within the acceptable control error, it be sent to the emergency handler.

↑ 1 👍 1 2 replies

xmfcx on Sep 20, 2022 Maintainer Author

If the bounding box of the vehicle is tight, even slight deviations from the center line can cause not checking the collisions. So this error can cause issues even if tracking error is low.

👍 1

yukkysaito on Nov 25, 2022 Maintainer

> If the bounding box of the vehicle is tight, even slight deviations from the center line can cause not checking the collisions. So this error can cause issues even if tracking error is low.

Yes.
My idea is as follows

1. Include a tolerance for position error as an output of planning (trajectory msg). Within the tolerance, it is guaranteed that there will be no collisions with obstacles.
2. The control module will raise an error if the control error is not within the tolerance. In some cases, the system will stop.

TakaHoribe on Sep 23, 2022 Maintainer

@xmfcx As discussed above, the idea behind the current design is that the planning module checks a collision on the desired path with a certain margin, and the ego vehicle should stop if the tracking error is over the margin. (The reason the path is not generated from the ego pose is to provide a stable reference to the controller.)

But I don't think the current approach is a good way. It is too restrictive, especially for not well-tuned controllers.
The requirement is that "even if the control is shaky and deviates from the reference path, the vehicle must be able to drive as long as it is safe". We don't want to care how long the tracking deviation is.

Here is some suggestion.

- In the planning module, calculate the path from the ego, and to perform collision ckeck to the path.
    - This method can ensure a certain degree of safety while keeping planning-control loose coupling.
    - However, since the route is calculated without knowing the controller's logic, it is not accurate that much.
- The planning module checks the collision for the path calculated by the control module, like predicted path of the MPC.
    - This method can check the future trajectory of the ego vehicle more accurately.
    - However, as discussed above, some controllers like pure_pursuit cannot calculate the predicted trajectory (although it can be forced to do so by simulation).

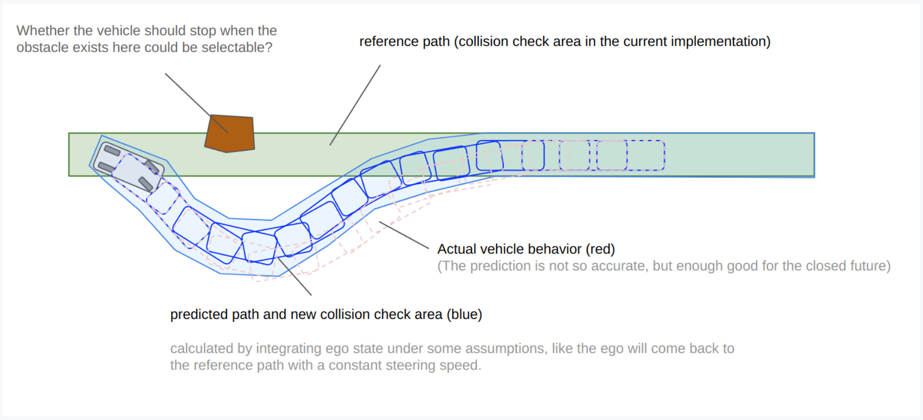
I suggest going with the first one. How to predict the future path without knowing the controller is, I think just integrating the current steering and velocity is enough for the first implementation.

(The ultimate way is to combine planning and control. If control commands can be generated simultaneously with path generation, we do not have to worry about this mismatch between ideal planning and actual behavior. But it is difficult from a modularity standpoint. We need to give up some accuracy or performance to maintain the clean architecture.)

↑ 1 3 replies

TakaHoribe on Sep 23, 2022 Maintainer edited

This is a rough sketch.


Whether the vehicle should stop when the obstacle exists here could be selectable?
reference path (collision check area in the current implementation)
Actual vehicle behavior (red)
(The prediction is not so accurate, but enough good for the closed future)
predicted path and new collision check area (blue)
calculated by integrating ego state under some assumptions, like the ego will come back to the reference path with a constant steering speed.

xmfcx on Sep 23, 2022 Maintainer Author

https://github.com/orgs/autowarefoundation/discussions/2859#discussioncomment-3633659 Here I've shared my solution which is similar to your first option.

Basically we still give the green path as reference to the controller.

But calculate blue one for collision checking purposes.

TakaHoribe on Sep 23, 2022 Maintainer

I personally agree with your approach. It would be also good to have an option to check which path the collision check is performed for. Or maybe both. Like this.

```
collision_check_reference_path: true
collisiton_check_predicted_path: true
```

👍 2

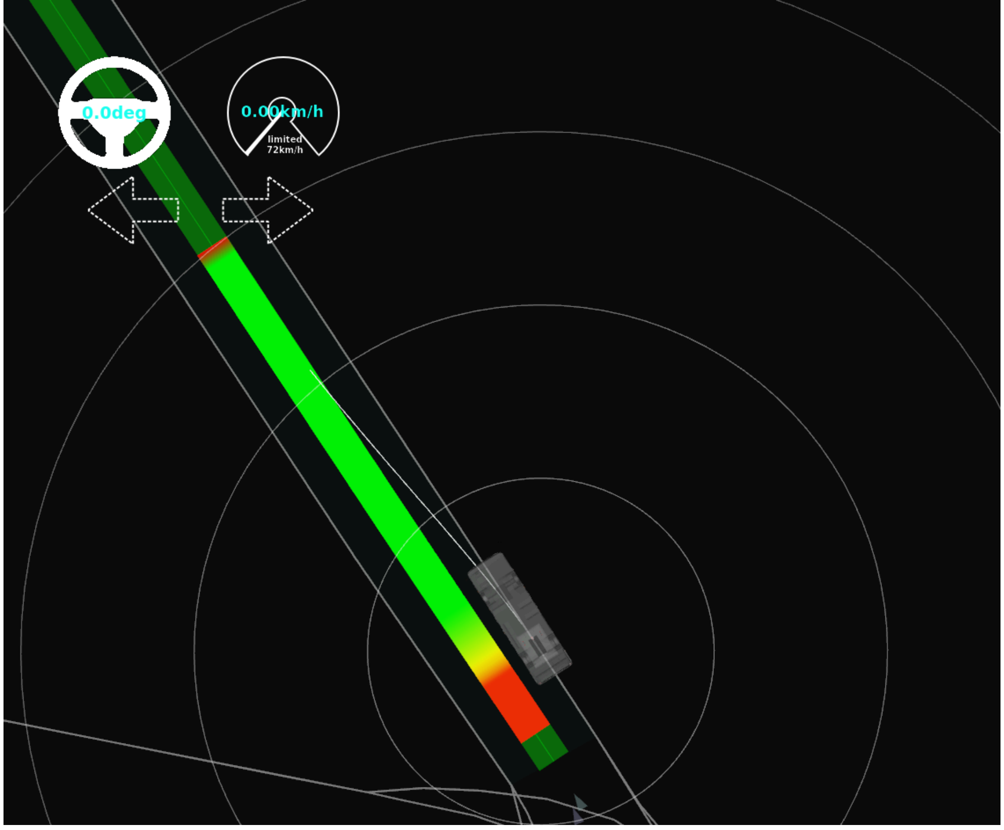
brkay54 on Dec 5, 2022 Collaborator

Hello everyone, I want to make a suggestion for control module design.

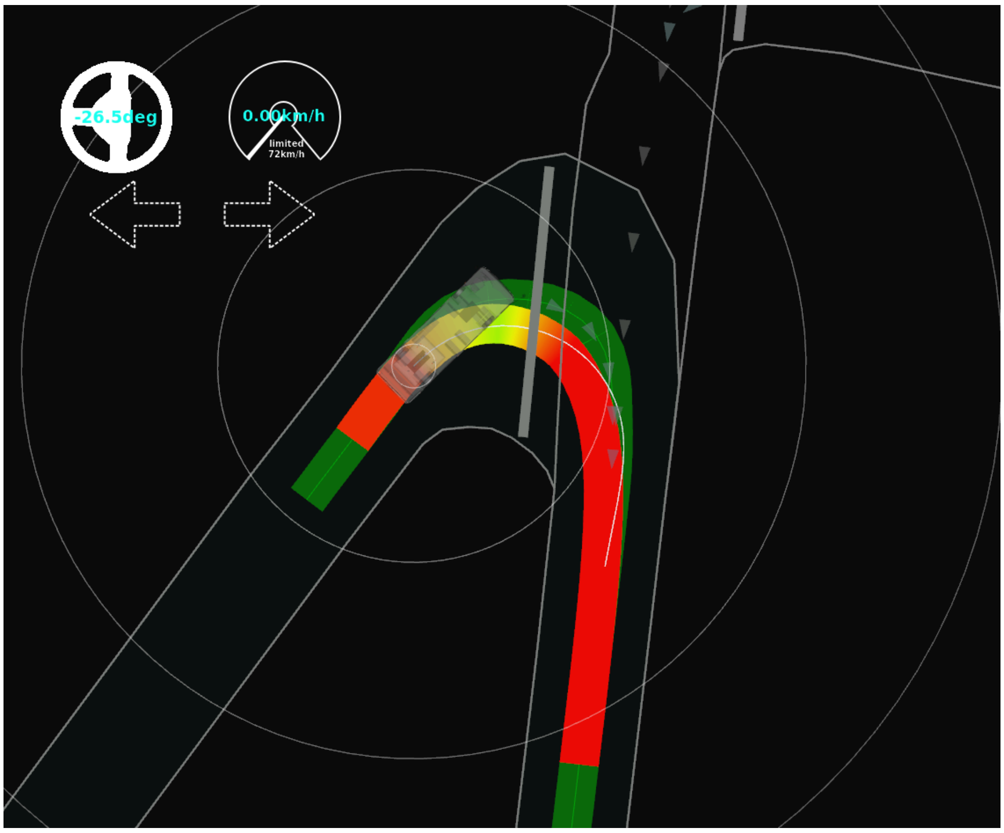
In current implementation of Autoware, control module is not responsible for collision checking, except
obstacle_collision_checker package (It can stop the vehicle just for emergency situations.). Planning module is making
stop planning and collision checking using the reference trajectory.

However, vehicles actual path would be apart from reference trajectory in some cases such as **starting driving with high
lateral error** and at **sharp turns**.

Starting driving with high lateral error:



Vehicle in sharp turns:

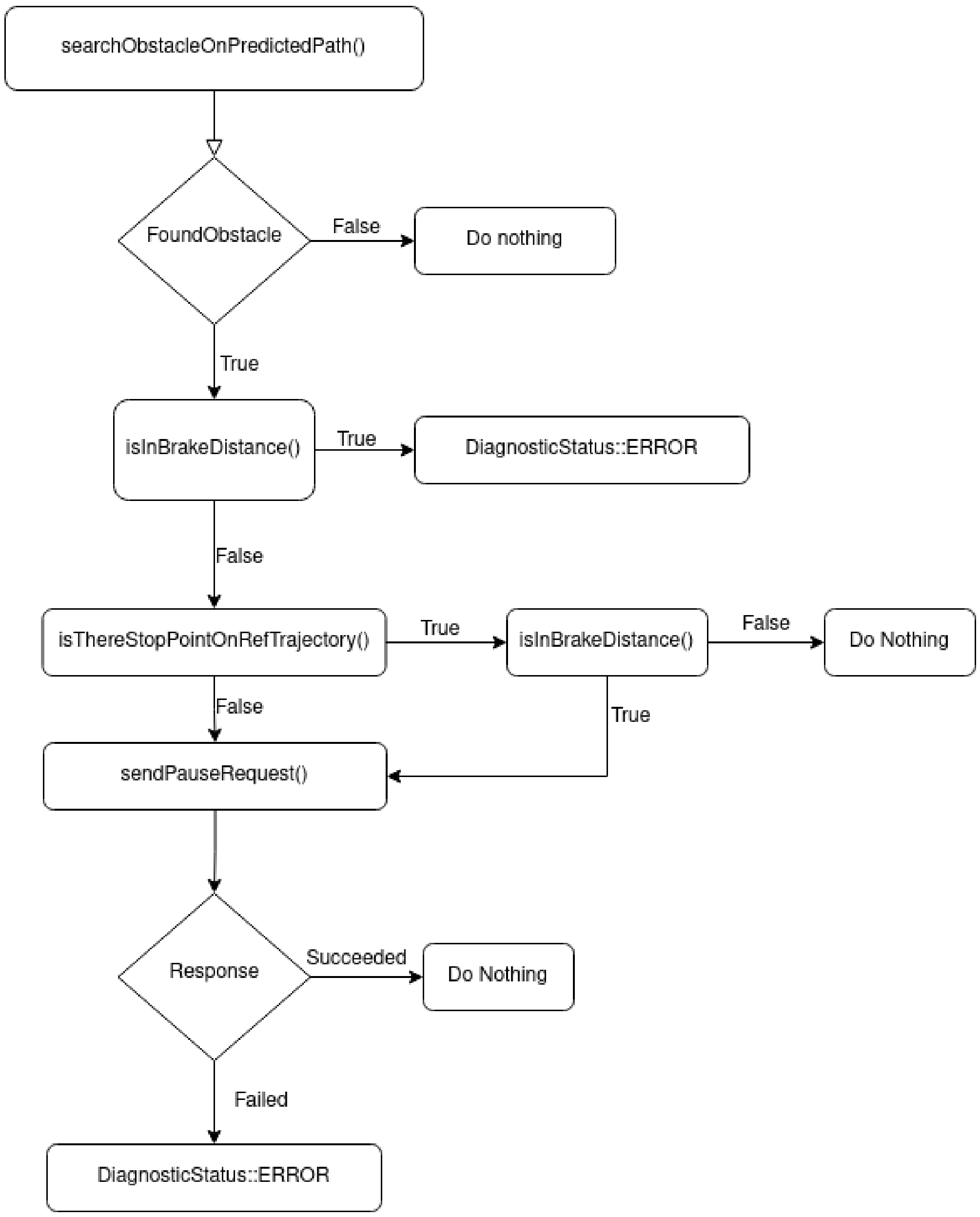


NOTE: White lines are the predicted path of vehicle

In this scenarios, planning may not handle the collision in some part of the path because it does collision checking just
on reference trajectory. I think this is a critical safety issue, because vehicle will be open to collision in some cases.

Because vehicle's actual path is strictly depend on the controller, I want to suggest that we should add a node (or we can
refactor the obstacle_collision_checker package to add new functionalities) to check obstacle collision on predicted path
and stop the vehicle if the collision was not handled by the planning module. To make stopped the vehicle, node will
use the pause interface (in vehicle_cmd_gate).

Here is node's flowchart:



This node is going to stop the vehicle just for collisions planning can not handle.

Pros:

- easy to implement
- modularity of planning and control still can be provided
- more accurate path for collision checking
- less false positive collisions

Cons:

- It depends controller's predicted path

All controllers in autoware can create predicted path. However, this will make creating predicted trajectory mandatory
for new controllers. I think it is not a big problem because even the most basic controller can create predicted path by
using kinematic model of vehicle iteratively (like in pure_pursuit).

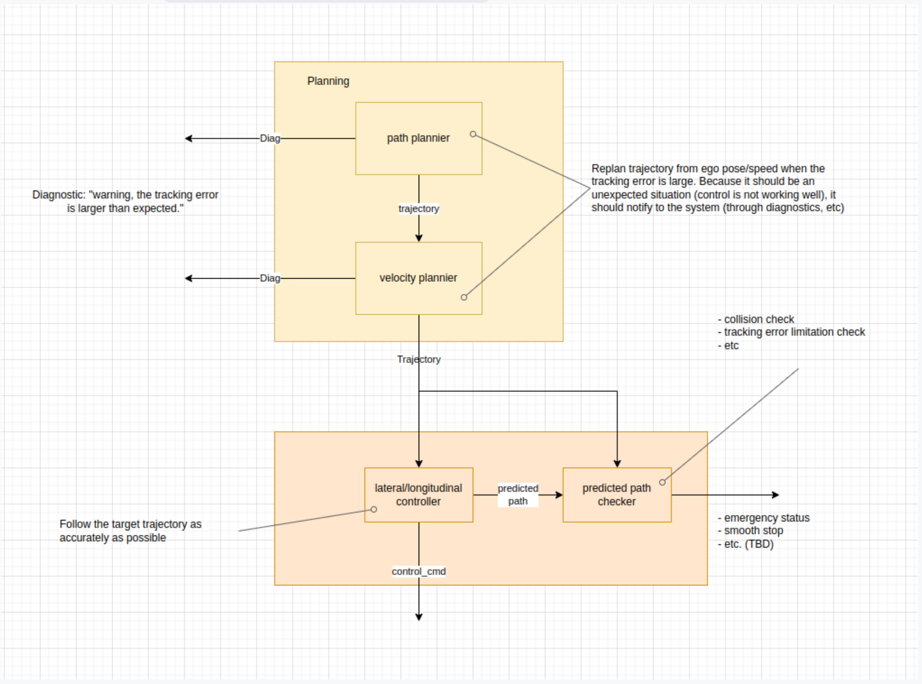
2

9 replies

Show 4 previous replies

TakaHoribe on Dec 7, 2022 Maintainer

The overall system is like this. The proposed node from **@brkay54** is written as a `predicted_path_checker` in this figure



Planning
path plannier
Diag
trajectory
velocity plannier
Diag
Diagnostic: "warning, the tracking error is larger than expected."
Replan trajectory from ego pose/speed when the tracking error is large. Because it should be an unexpected situation (control is not working well), it should notify to the system (through diagnostics, etc)
Trajectory
- collision check
- tracking error limitation check
- etc
lateral/longitudinal controller
predicted path
predicted path checker
Follow the target trajectory as accurately as possible
- emergency status
- smooth stop
- etc. (TBD)
control_cmd

brkay54 on Dec 8, 2022 Collaborator

@TakaHoribe Thank you for your feedback! I agree with you except this:

> Because the pause API is (will be) widely used even in the external application, anyone can cancel the pause sent by this module. Since this function is safety-critcal, it should be canceled easily. I suggest to use the smooth stop command as used in this PR for MRM.

I agree with you, using pause_interface causes some safety problems. However, using mrm_comfortable_stop_operator causes changing velocity on trajectory. I think, it can cause conflict between planning and control modules like you said in this PR .

For video: https://user-images.githubusercontent.com/45468306/203044604-08a24063-2cf5-4ad1-9136-b5891f482539.mp4

I think we can make some changes in pause_interface to avoid problems you mentioned. For example, we can prioritize the SetPause requests or we can declare another pause_interface for only control modules and we can make it prior to other interface.

mehmetdogru on Dec 9, 2022 Maintainer

@TakaHoribe

Thank you for your inputs.

> I suggest implementing a similar approach in the path planning (obstacle_avoidance_planner)

Isn't this approach already implemented in `obstacle_avoidance_planner` ? If this is what you meant; for smaller values for `max_ego_moving_dist_for_replan` -converges on traj from base_link- does not work quite well.

brkay54 on Dec 19, 2022 Collaborator

I created PR for `predicted_path_checker`. However, I am not sure to how we make the vehicle stop. If there is no more opinion, I am going to make some changes on `pause_interface` (I will share detailed suggestion later).

mitsudome-r on Dec 22, 2022 Maintainer

Conclusion from meeting between @mehmetdogru , @brkay54 , @TakaHoribe:

- Whenever `predicted_path_checker` detects obstacle on predicted path, it will send pause command through `pause_interface`. It will remove the pause command if obstacle disappears from the predicted path.
- However, since `pause_interface` can be overridden by other modules, `predicted_path_checker` will send emergency state to trigger emergency stop if urgent stop is needed (e.g., if ego is too close to the obstacle). This will not be recovered without human operation.