Status Report

Project Path Planning Team1

Report Period:

CW 47

Milestone:

Milestone	Date	Status	
Run first successful PythonRobotics Path Planning Algorithm with CoppeliaSim	08.11.2023	☺	
Run first successful PythonRobotics Path Tracking Algorithm with CoppeliaSims (Open Loop)	15.11.2023	☺	
Raspberry Pi Software Deployment	15.11.2023	☺	
Run first successful PythonRobotics Path Tracking Algorithm with CoppeliaSims (Closed Loop)	22.11.2023	☺	
Simulation: Path Planning - Car Model Adaption	29.11.2023	☹	
Simulation: Path Tracking - Car Model Adaption	29.11.2023	☹	
SW Integration for real RVR	13.12.2023	☹	
Path Planning/Tracking Runtime Optimization	03.01.2024	☹	
Complete Documentation	31.01.2024	☹	
		☺ 😐 ☹	

Results in the report period

Results (achieved, not achieved, planned)	
Results achieved	**Software Tasks (assigned to Laurens):** • **Implementation of closed loop Path Tracking Algorithm** **Deployment Tasks (assigned to Laurens):** • **Deploy Path Planning SW to RaspberryPi** **Environment Modeling Tasks (assigned to Roman):** • **CoppeliaSim: Model car/robot (Model decision by Car Hardware Team)** • **CoppeliaSim: Model common parking scenes (Parallel Parking, Perpendicular Parking)** • **Modify the car model with different caterpillars that have a script for movement.** • **Make a model with the dimensions of a real car.** **Documentation Tasks (assigned to Roman):** • **Documentation of Issues and Solvings** **Deployment Tasks:** • **Runtime and PathLength Profiling SW for RaspberryPi (assigned to Lam)**

Results not achieved	
Planned results fort he next period	POSTPONED TOPICS: • Build OMPL Python binding for C++ (Windows & MacOS) • Use OMPL Path-Planning-Algorithms in CoppeliaSim

Problems, Risks, Measures in Report Period

a) Which problems have been occured?

Path Tracking / Path Planning:

- Use Coppelia “simulation time” instead of “system time” to compute time delta in Stanley Controller
- Use CoppeliaSim Real-Time Mode:

- API **sim.setJointTargetVelocity(handleID, velocity=1.0) doesn’t set velocity to 1 [deg/s]** Instead the velocity of robot joints is multiplied by **factor 57.29578**
- Increase robot radius in Path Planning Dijkstra / AStar to avoid collisions with rectangular obstacles
  → Path Planning Algorithm for Autonomous Parking needed which can handle rectangular obstacles

b) Which (new) risks can lead to problems?

- **Runtime of Motion Planning algorithms**

c) So far undertaken countermeasures? Who? Until when?

- **Runtime Analysis on Raspberry Pi (assigned to Lam)**
- **Runtime Optimization Planning/Tracking (assigned to tbd)**

d) Necessary decisions to take? By whom? Until when?

-
-

Issues

Issues	**Name Model parking scenes**	**Category Environment Modeling Tasks**
Issues appeared	The script is not working, all videos and documentations are too old for this version of CoppeliaSim	
What we tried	1) Watched 03: Path Planning with a Differential Drive Robot V-Rep/CoppeliaSim | Tutorial (03: Path Planning with a Differential Drive Robot V-Rep/CoppeliaSim | Tutorial) - version used in this video is too old and there is no Path planning module in CoppeliaSim now. 2) Watched CoppeliaSim: Differential Drive Car, Control (2 of 3) (CoppeliaSim: Differential Drive Car, Control (2 of 3)) - the script is not working on our model (maybe because of 4 wheels instead of 2) 3) Watched CoppeliaSim: Line Follower Car, Vision sensor (all parts) (CoppeliaSim: Line Follower Car, Vision sensor (part 2 of 4)) - we will try to do it on 15.11	
Solutions	Choose other caterpillars that have a script (model browser -> components -> locomotion and propulsion).	

Issues	**Name Car collision**	**Category Environment Modeling Tasks**
Issues appeared	The caterpillars have some invisible dynamic cuboids that are used for moving. Because of them, the car collides with air and rides in the air.	
What we tried	Make cuboids smaller (Decreases speed of the car); Move them to the center of the car (But they start rotating with bigger radius); Remove them (The car just stops moving);	

Solutions	Moved the joints inside the car. Make these cuboids smaller, so now the car can ride on the floor properly, but the speed of the car decreases.

Issues	**Name**	**Category**
Issues appeared		
What we tried		
Solutions		

Stanley Controller (Stanford University - DARPA Grand Challenge)

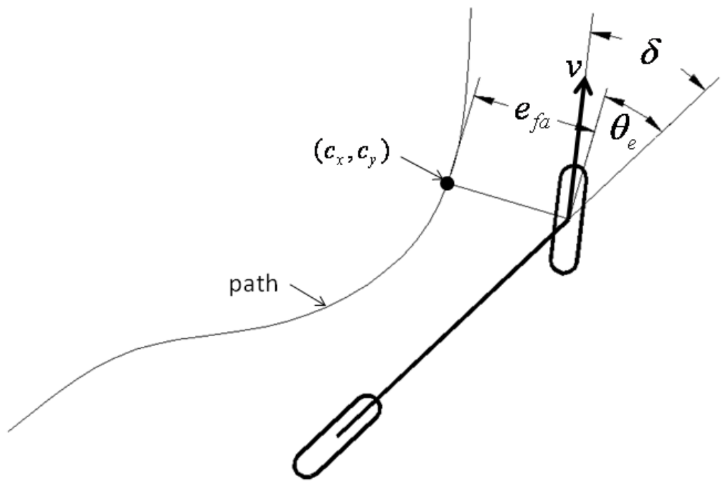

Figure 14: Stanley method geometry

= Nonlinear feedback function of the cross track error.
= Controller continuously computes error to path to adjust the steering angle accordingly, aiming to minimize the deviation and keep the vehicle on track
= Has exponential convergence

Variable	Description
(c_x, c_y)	Nearest path point
v	Vehicle velocity
e_fa	Cross track error (lateral deviation of the vehicle from the desired path) measured from the center of the front axle to the nearest path point
δ	Front axle steering angle
θ	Heading of the vehicle
θ_p	Heading of the path at closest point (c_x, c_y)
k	Gain parameter

Steering error θ_e between Heading of vehicle and Heading of path:

$$\theta_e = \theta - \theta_p,$$

Term 1: Steering error to get parallel to path heading
Term 2: Steering error to converge toward the path

$$\delta(t) = \theta_e(t) + \tan^{-1}\left(\frac{ke_{fa}(t)}{v_x(t)}\right)$$

If e_fa is non-zero → Steering angle delta is adjusted so that the intended trajectory intersects the path tangent from (c_x, c_y) at kv(t) untis from the front axle. As e_fa increases, the wheels are steered further toward the path.

```python
def stanley_control(state, cx, cy, cyaw, last_target_idx):
    """
    Stanley steering control.

    :param state: (State object)
    :param cx: ([float])
    :param cy: ([float])
    :param cyaw: ([float])
    :param last_target_idx: (int)
    :return: (float, int)
    """
    current_target_idx, error_front_axle = calc_target_index(state, cx, cy)

    if last_target_idx >= current_target_idx:
        current_target_idx = last_target_idx

    # theta_e corrects the heading error
    theta_e = normalize_angle(cyaw[current_target_idx] - state.yaw)
    # theta_d corrects the cross track error
    theta_d = np.arctan2(k * error_front_axle, state.v)
    # Steering control
    delta = theta_e + theta_d

    return delta, current_target_idx
```